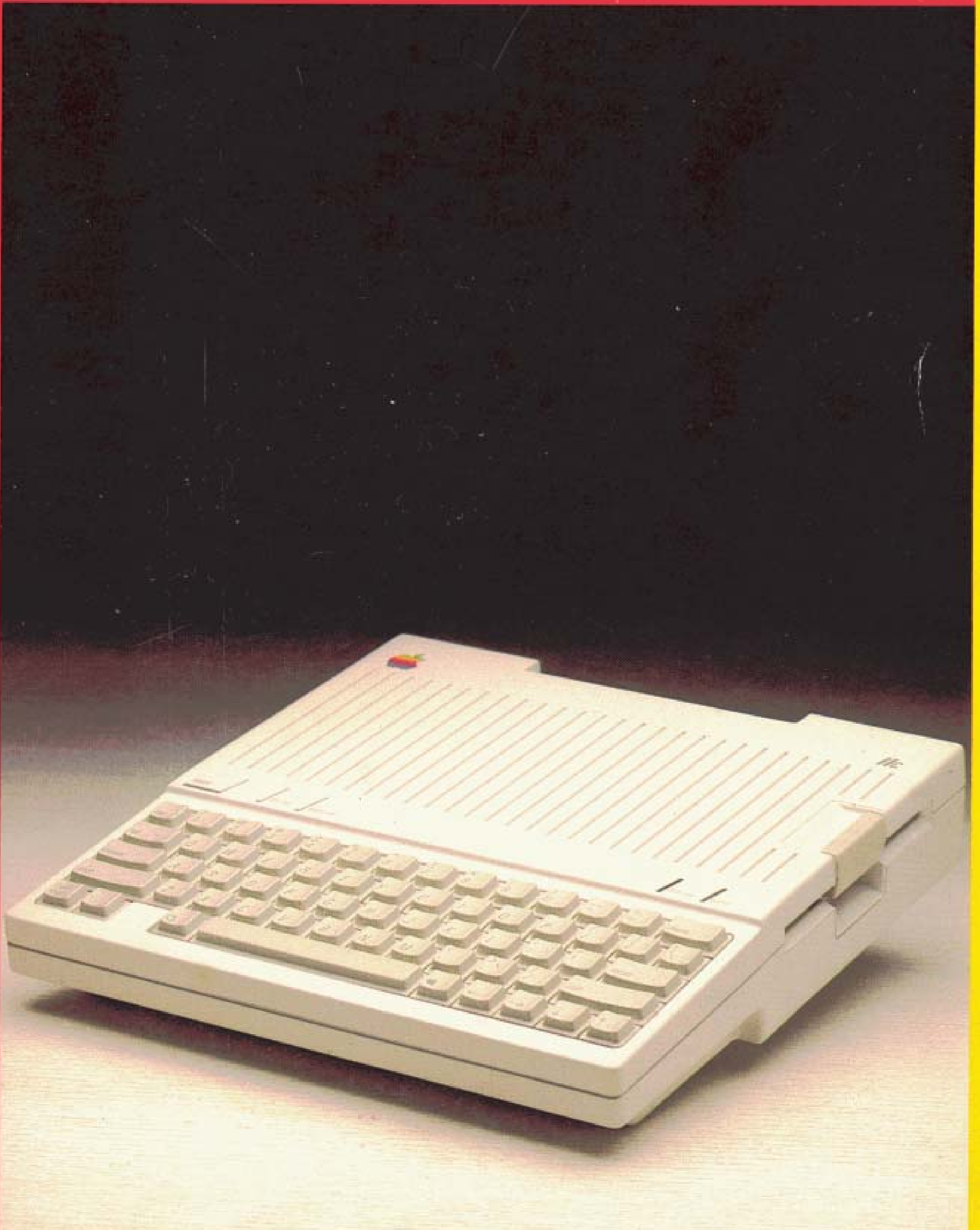
The Apple IIc Reference Manual

Volume 2

The Apple IIc



Customer Satisfaction

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable disks and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

Limitation on Warranties and Liability

Even though Apple has tested the software described in the manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

This manual and the software (computer programs) described in it are copyrighted by Apple or by Apple's software suppliers, with all rights reserved. Under the copyright laws, this manual or the programs may not be copied, in whole or part, without the written consent of Apple, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose. For some products, a multi-use license may be purchased to allow the software to be used on more than one computer owned by the purchaser, including a shared-disk system. (Contact your authorized Apple dealer for information on multi-use licenses.)

Product Revisions

Apple cannot guarantee that you will receive notice of a revision to the software described in the manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple dealer.

© Apple Computer, Inc. 1984
20525 Mariani Avenue
Cupertino, California 95014

Warning

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

Reorder Apple Product #A2L4030

Apple IIc Reference Manual
Volume 2

The Apple IIc

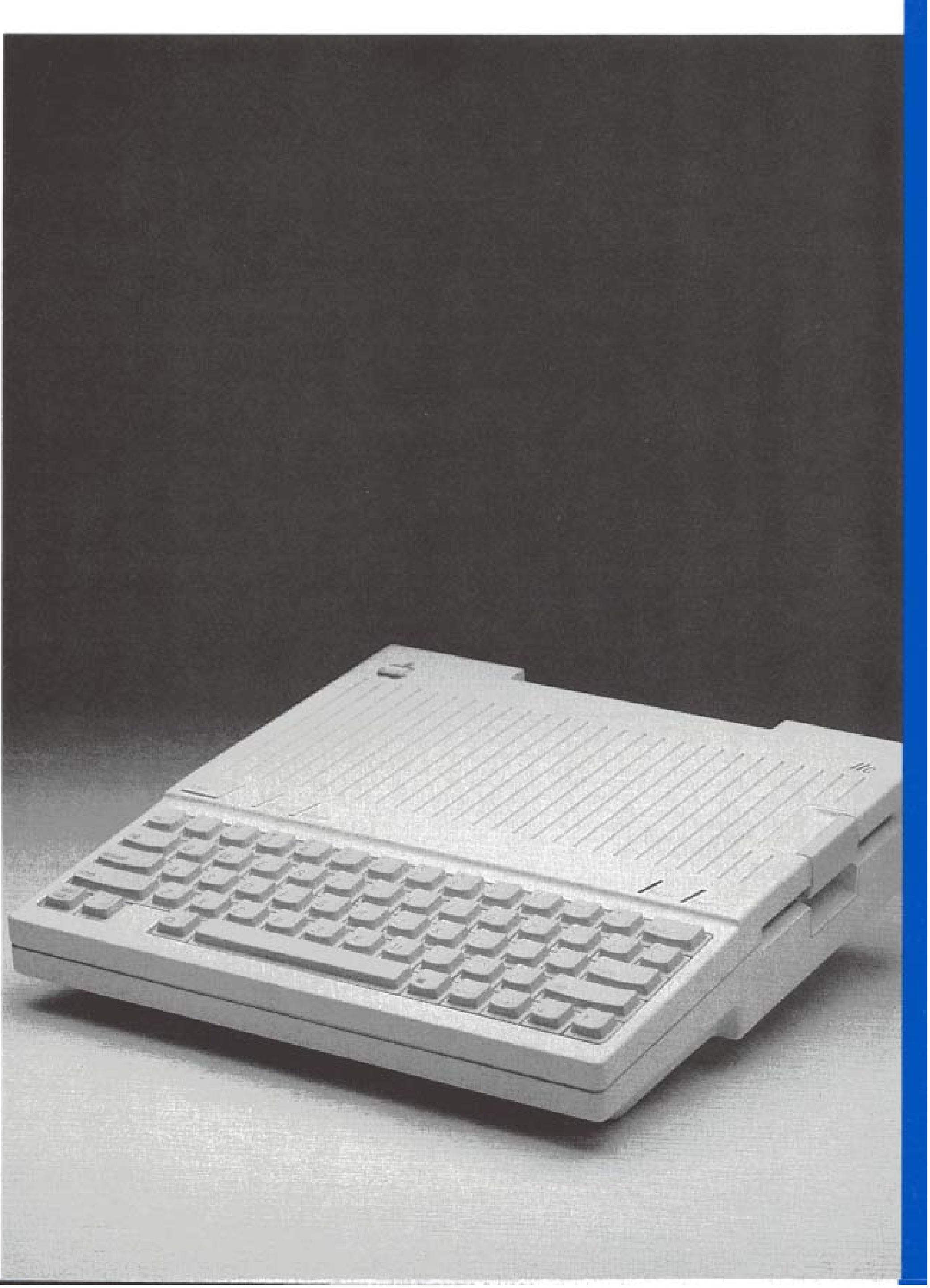


Table of Contents

Appendix D Operating Systems and Languages 37

Appendix E Interrupts 41

List of Figures and Tables

Appendix A The 65C02 Microprocessor

Appendix B Memory Map

Appendix C Important Firmware Locations

Appendix E Interrupts

Appendix F Apple II Series Differences

Appendix G USA and International Models

Appendix H Conversion Tables

Preface

This volume, Volume 2 of the *Apple IIc Reference Manual*, contains nine appendixes, a bibliography, and a glossary.

Appendix A contains a description of the differences between the 6502 and the 65C02 microprocessors, plus a reprint of the manufacturer's data sheet for the 65C02 microprocessor.

Appendixes B and C contain tables of the important RAM, ROM, and hardware addresses in the Apple IIc. The reader can use these tables to find locations by address, the index to find them by label, the firmware listings to find them as defined and used, and the chapters to find them described in the context of their function.

Appendix B is a memory map of the Apple IIc, including detailed tables of page zero, page three, the screen holes, and the hardware page.

Appendix C lists the *published* firmware entry points, arranged by address, and indicates where in the manual they are described. The list includes I/O firmware (pages $C1 through $CF) and Monitor firmware (pages $F0 through $FF). For Applesoft interpreter firmware (pages $D0 through $EF), refer to the *Applesoft BASIC Programmer's Reference Manual*, Volumes 1 and 2.

Appendix D discusses what operating systems and languages run on the Apple IIc, and what features they do and do not use.

Appendix E describes how to use the Apple IIc's interrupt handling capabilities.

Appendix F contains an overview of the differences among the Apple II series computers.

Appendix G contains the keyboard layouts, code conversion tables, and external power supply characteristics of USA and international models of the Apple IIc.

Appendix H contains reference tables for code and number base conversion.

Appendix I contains a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc. The listings do not include the built-in Applesoft interpreter, which is discussed in the *Applesoft BASIC Programmer's Reference Manual.*

The Bibliography lists articles and books containing additional information about the Apple IIc and related products.

The Glossary defines many of the technical terms used in this manual.

The 65C02 Microprocessor

appendix a

This appendix contains a description of the differences between the 6502 and the 65C02 microprocessor. It also contains the data sheet for the NCR 65C02 microprocessor.

In the data sheet tables, execution times are specified in number of cycles. One cycle time for the Apple IIc equals 0.978 microseconds.

If you want to write programs that execute on all computers in the Apple II series, make sure your code uses only the subset of instructions present on the 6502.

A.1 Differences Between 6502 and 65C02

The data sheet lists the new instructions and addressing modes of the 65C02. This section supplements that information by listing the instructions whose execution times or results have changed.

A.1.1 Differing Cycle Times

In general, differences in execution times are significant only in time-dependent code, such as precise wait loops. Fortunately, instructions with changed execution times are few.

Table A-1 lists the instructions whose number of instruction execution cycles on the 65C02 is different from the number on the 6502.

Table A-1. *Cycle Time Differences*

Instruction/Mode	Opcode	6502 Cycles	65C02 Cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

A.1.2 Differing Instruction Results

It is important to note that the BIT instruction when used in immediate mode (code $89) leaves Processor Status Register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the Status Register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6. However, all BIT instructions on both versions of the processor set status bit 1 (Z) if the memory location contained a zero.

Also note that if the JMP indirect instruction (code $6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP ($2FF) gets ADL from location $2FF on both processors. But on the 65C02, ADH comes from $300; on the 6502, ADH comes from $200.

A.2 Data Sheet

The remaining pages of this appendix are copyright 1982, NCR Corporation, Dayton, Ohio, and are reprinted with their permission.

NCR

NCR65C02

▪ GENERAL DESCRIPTION

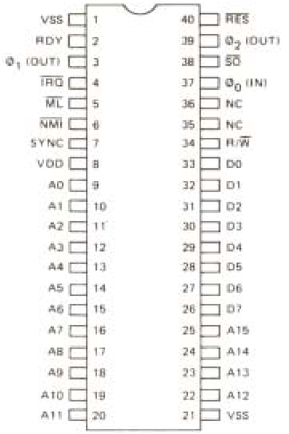
The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

▪ FEATURES

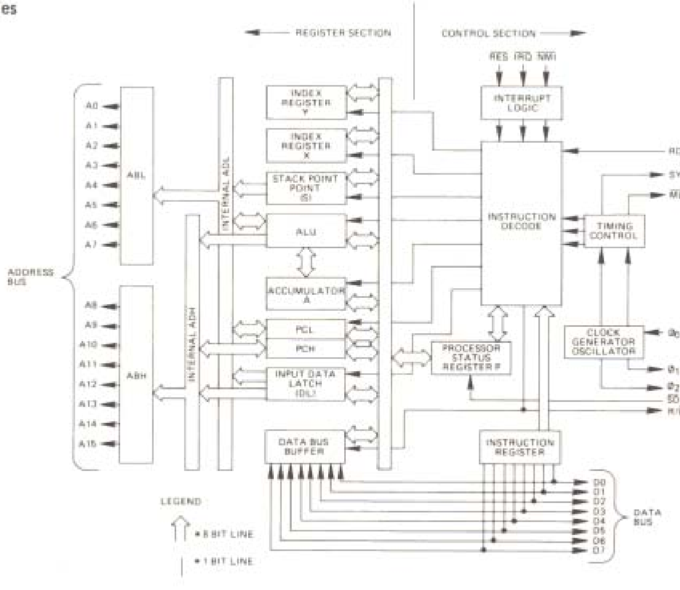
- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 Hz for even lower power consumption (pseudo-static: stop during Ø_2 high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for ($\overline{\text{RDY}}$, $\overline{\text{IRQ}}$, $\overline{\text{SO}}$, $\overline{\text{NMI}}$ and $\overline{\text{RES}}$)

* Specifications are subject to change without notice.

▪ PIN CONFIGURATION



▪ NCR65C02 BLOCK DIAGRAM



NCR65C02

▪ ABSOLUTE MAXIMUM RATINGS: (V_{DD} = 5.0 V ± 5%, V_{SS} = 0 V, T_A = 0° to + 70°C)

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{DD}	−0.3 to +7.0	V
INPUT VOLTAGE	V_{IN}	−0.3 to +7.0	V
OPERATING TEMP.	T_A	0 to + 70	°C
STORAGE TEMP.	T_{STG}	−55 to + 150	°C

▪ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
$\overline{\text{IRQ}}$ *	Interrupt Request
RDY *	Ready
$\overline{\text{ML}}$	Memory Lock
$\overline{\text{NMI}}$ *	Non-Maskable Interrupt
SYNC	Synchronize
$\overline{\text{RES}}$ *	Reset
$\overline{\text{SO}}$ *	Set Overflow
NC	No Connection
R/$\overline{\text{W}}$	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
$Ø_0$	Clock Input
$Ø_1$, $Ø_2$	Clock Output

*This pin has an optional internal pullup for a No Connect condition.

▪ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX	UNIT
Input High Voltage $Ø_0$ (IN)	V_{IH}	V_{SS} + 2.4	–	V_{DD}	V
Input High Voltage $\overline{\text{RES}}$, $\overline{\text{NMI}}$, RDY, $\overline{\text{IRQ}}$, Data, S.O.		V_{SS} + 2.0	–	–	V
Input Low Voltage $Ø_0$ (IN)	V_{IL}	V_{SS} −0.3	–	V_{SS} + 0.4	V
$\overline{\text{RES}}$, $\overline{\text{NMI}}$, RDY, $\overline{\text{IRQ}}$, Data, S.O.		–	–	V_{SS} + 0.8	V
Input Leakage Current (V_{IN} = 0 to 5.25V, V_{DD} = 5.25V) With pullups	I_{IN}	-30	–	+30	μA
Without pullups		–	–	+1.0	μA
Three State (Off State) Input Current (V_{IN} = 0.4 to 2.4V, V_{CC} = 5.25V) Data Lines	I_{TSI}	–	–	10	μA
Output High Voltage (I_{OH} = −100 μAdc, V_{DD} = 4.75V SYNC, Data, A0-A15, R/W)	V_{OH}	V_{SS} + 2.4	–	–	V
Out Low Voltage (I_{OL} = 1.6mAdc, V_{DD} = 4.75V SYNC, Data, A0-A15, R/W)	V_{OL}	–	–	V_{SS} + 0.4	V
Supply Current f = 1MHz	I_{DD}	–	–	4	mA
Supply Current f = 2MHz	I_{DD}	–	–	8	mA
Capacitance (V_{IN} = 0, T_A = 25°C, f = 1MHz)	C				pF
Logic	C_{IN}	–	–	5	
Data		–	–	10	
A0-A15, R/W, SYNC	Cout	–	–	10	
$Ø_0$ (IN)	$CØ_0$ (IN)	–	–	10	

▪ AC CHARACTERISTICS V_{DD} = 5.0V ± 5%, T_A = 0°C to 70°C, Load = 1 TTL + 130 pF

		1MHZ		2MHZ		3MHZ		
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Unit
Delay Time, $Ø_0$ (IN) to $Ø_2$ (OUT)	t_{DLY}	–	60	–	60	20	60	nS
Delay Time, $Ø_1$ (OUT) to $Ø_2$ (OUT)	t_{DLY1}	–20	20	–20	20	–20	20	nS
Cycle Time	t_{CYC}	1.0	5000*	0.50	5000*	0.33	5000*	µS
Clock Pulse Width Low	t_{PL}	460	–	220	–	160	–	nS
Clock Pulse Width High	t_{PH}	460	–	220	–	160	–	nS
Fall Time, Rise Time	t_F, t_R	–	25	–	25	–	25	nS
Address Hold Time	t_{AH}	20	–	20	–	0	–	nS
Address Setup Time	t_{ADS}	–	225	–	140	–	110	nS
Access Time	t_{ACC}	650	–	310	–	170	–	nS
Read Data Hold Time	t_{DHR}	10	–	10	–	10	–	nS
Read Data Setup Time	t_{DSU}	100	–	60	–	60	–	nS
Write Data Delay Time	t_{MDS}	–	30	–	30	–	30	nS
Write Data Hold Time	t_{DHW}	20	–	20	–	15	–	nS
$\overline{SO}$ Setup Time	t_{SO}	100	–	100	–	100	–	nS
Processor Control Setup Time**	t_{PCS}	200	–	150	–	150	–	nS
SYNC Setup Time	t_{SYNC}	–	225	–	140	–	100	nS
$\overline{ML}$ Setup Time	t_{ML}	–	225	–	140	–	100	nS
Input Clock Rise/Fall Time	$t_{FØo}$,$t_{RØo}$	–	25	–	25	–	25	nS

*NCR65C02 can be held static with $Ø_2$ high.

**This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

▪ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

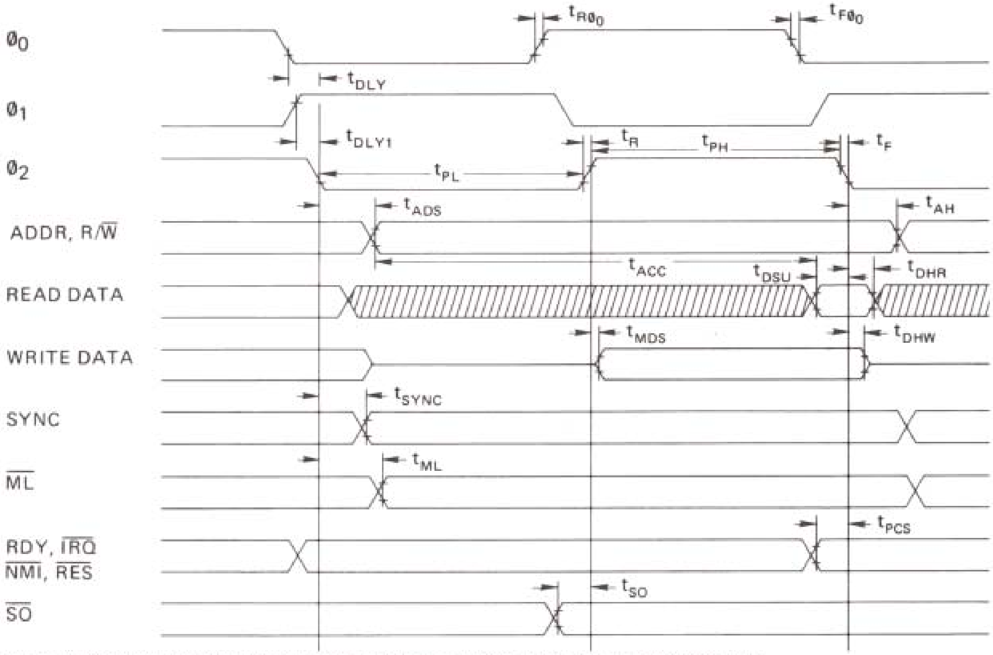
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code — Bytes — Cycles X2 — 2 — 2 X3, X7, XB, XF — 1 — 1 44 — 2 — 3 54, D4, F4 — 2 — 4 5C — 3 — 8 DC, FC — 3 — 4
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.

▪ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during $Ø_2$.
Unused input-only pins ($\overline{IRQ}$, $\overline{NMI}$, RDY, $\overline{RES}$, $\overline{SO}$).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to V_{DD} (approximately 250 K ohm.)

NCR65C02

■ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

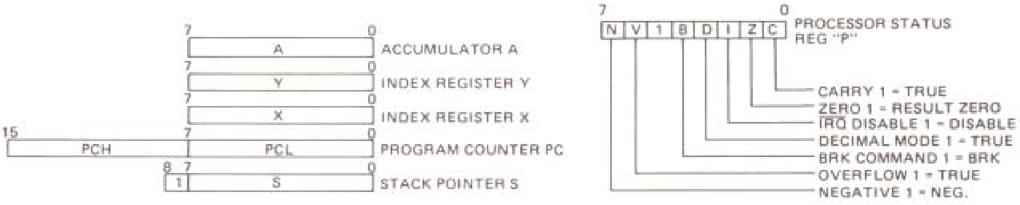
■ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG,X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

■ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND,X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

▪ MICROPROCESSOR PROGRAMMING MODEL



▪ FUNCTIONAL DESCRIPTION

Timing Control
The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

Program Counter
The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

Instruction Register and Decode
Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

Arithmetic and Logic Unit (ALU)
All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

Accumulator
The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

Index Registers
There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

Stack Pointer
The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts ($\overline{NMI}$ and $\overline{IRQ}$). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

Processor Status Register
The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

NCR65C02

▪ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

Implied Addressing [Implied]
In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Accumulator Addressing [Accum]
This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

Immediate Addressing [Immediate]
With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

Absolute Addressing [Absolute]
For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

Zero Page Addressing [Zero Page]
Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

Absolute Indexed Addressing [ABS, X or ABS, Y]
Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

Zero Page Indexed Addressing [ZPG, X or ZPG, Y]
Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

Relative Addressing [Relative]
Relative addressing is used only with branch instructions; it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is −128 to +127 bytes from the next instruction.

Zero Page Indexed Indirect Addressing [(IND, X)]
With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

***Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**
With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

Indirect Indexed Addressing [(IND), Y]
This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

***Zero Page Indirect Addressing [(ZPG)]**
In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

Absolute Indirect Addressing [(ABS)] (Jump Instruction Only)
The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: * = New Address Modes

▪ SIGNAL DESCRIPTION

Address Bus (A0-A15)
A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

Clocks ($\emptyset_0$, $\emptyset_1$, and $\emptyset_2$)
$\emptyset_0$ is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The $\emptyset_2$ clock output is in phase with $\emptyset_0$. The $\emptyset_1$ output pin is 180° out of phase with $\emptyset_0$. (See timing diagram.)

Data Bus (D0-D7)
The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

Interrupt Request ($\overline{\text{IRQ}}$)
This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The $\overline{\text{IRQ}}$ is sampled during $\emptyset_2$ operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during $\emptyset_1$. The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further $\overline{\text{IRQs}}$ may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

Memory Lock ($\overline{\text{ML}}$)
In a multiprocessor system, the $\overline{\text{ML}}$ output indicates the need to defer the rearbitration of the next bus cycle to ensure the integrity of read-modify-write instructions. $\overline{\text{ML}}$ goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

Non-Maskable Interrupt ($\overline{\text{NMI}}$)
A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The $\overline{\text{NMI}}$ is sampled during $\emptyset_2$; the current instruction is completed and the interrupt sequence begins during $\emptyset_1$. The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another $\overline{\text{NMI}}$ can occur before the first is finished. Care should be taken when using $\overline{\text{NMI}}$ to avoid this.

Ready (RDY)
This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one ($\emptyset_1$), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two ($\emptyset_2$) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

Reset ($\overline{\text{RES}}$)
This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after V_{DD} reaches operating voltage from a power down. A positive transistion on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on $\overline{\text{RES}}$.

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

Read/Write (R/$\overline{\text{W}}$)
This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

Set Overflow ($\overline{\text{SO}}$)
A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of $\emptyset_1$.

Synchronize (SYNC)
This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during $\emptyset_1$ of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the $\emptyset_1$ clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

▪ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift One Bit Left
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Memory Bits with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
*BRA	Branch Always
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
*DEA	Decrement Accumulator
DEC	Decrement by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-or" Memory with Accumulator
*INA	Increment Accumulator
INC	Increment by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
*PHX	Push Index X on Stack
*PHY	Push Index Y on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
*PLX	Pull Index X from Stack
*PLY	Pull Index Y from Stack
ROL	Rotate One Bit Left
ROR	Rotate One Bit Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Bit
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
*STZ	Store Zero in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
*TRB	Test and Reset Memory Bits with Accumulator
*TSB	Test and Set Memory Bits with Accumulator
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

Note: * = New Instruction

▪ MICROPROCESSOR OP CODE TABLE

S D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X			TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs		0
1	BPL rel	ORA ind, Y	ORA*† (zpg)		TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X		1
2	JSR abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI rel	AND ind, Y	AND*† (zpg)		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X		3
4	RTI	EOR ind, X				EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC rel	EOR ind, Y	EOR*† (zpg)			EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY*			EOR abs, X	LSR abs, X		5
6	RTS	ADC ind, X			STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs		6
7	BVS rel	ADC ind, Y	ADC*† (zpg)		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY*		JMP*† abs (ind, X)	ADC abs, X	ROR abs, X		7
8	BRA* rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs		8
9	BCC rel	STA ind, Y	STA*† (zpg)		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X		9
A	LDY imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS rel	LDA ind, Y	LDA*† (zpg)		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE rel	CMP ind, Y	CMP*† (zpg)			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX*			CMP abs, X	DEC abs, X		D
E	CPX imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ rel	SBC ind, Y	SBC*† (zpg)			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX*			SBC abs, X	INC abs, X		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: * = New OP Codes
Note: † = New Address Modes

▪ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

			IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM			IMPLIED			(IND, X)			(IND), Y			ZPG, X			ZPG, Y			ABS, X			ABS, Y			RELATIVE			(ABS)			ABS (IND, X)			(ZPG)			PROCESSOR STATUS CODES	
MNE	OPERATION		OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	7 6 5 4 3 2 1 0 N V . B D I Z C	MNE
ADC	A + M + C → A	(1,3)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2				7D	4	3	79	4	3										72	5	2	N V . . . . Z C	ADC
AND	A ∧ M → A	(1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2				3D	4	3	39	4	3										32	5	2	N . . . . . Z .	AND
ASL	C ← 7 0 ← 0	(1)				0E	6	3	06	5	2	0A	2	1										16	6	2				1E	6	3																N . . . . . Z C	ASL
BCC	Branch if C=0	(2)																																		90	2	2										. . . .	BCC
BCS	Branch if C=1	(2)																																		B0	2	2										. . . .	BCS
BEQ	Branch if Z=1	(2)																																		F0	2	2										. . . .	BEQ
BIT	A ∧ M	(4,5)	89	2	2	2C	4	3	24	3	2													34	4	2				3C	4	3																M_7 M_6 . . . . Z .	BIT
BMI	Branch if N=1	(2)																																		30	2	2										. . . .	BMI
BNE	Branch if Z=0	(2)																																		D0	2	2										. . . .	BNE
BPL	Branch if N=0	(2)																																		10	2	2										. . . .	BPL
BRA	Branch Always	(2)																																		80	2	2										. . . .	BRA
BRK	Break														00	7	1																															. . . 1 . 1 . .	BRK
BVC	Branch if V=0	(2)																																		50	2	2										. . . .	BVC
BVS	Branch if V=1	(2)																																		70	2	2										 0	BVS
CLC	0 → C														18	2	1																															 0	CLC
CLD	0 → D														D8	2	1																															. . . . 0 . . .	CLD
CLI	0 → I														58	2	1																															. . . . . 0 . .	CLI
CLV	0 → V														B8	2	1																															. 0	CLV
CMP	A - M	(1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2				DD	4	3	D9	4	3										D2	5	2	N . . . . . Z C	CMP
CPX	X - M		E0	2	2	EC	4	3	E4	3	2																																					N . . . . . Z C	CPX
CPY	Y - M		C0	2	2	CC	4	3	C4	3	2																																					N . . . . . Z C	CPY
DEA	A - 1 → A											3A	2	1																																		N . . . . . Z .	DEA
DEC	M - 1 → M	(1)				CE	6	3	C6	5	2													D6	6	2				DE	6	3																N . . . . . Z .	DEC
DEX	X - 1 → X														CA	2	1																															N . . . . . Z .	DEX
DEY	Y - 1 → Y														88	2	1																															N . . . . . Z .	DEY
EOR	A ⊻ M → A		49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2				5D	4	3	59	4	3										52	5	2	N . . . . . Z .	EOR
INA	A + 1 → A											1A	2	1																																		N . . . . . Z .	INA
INC	M + 1 → M	(1)				EE	6	3	E6	5	2													F6	6	2				FE	6	3																N . . . . . Z .	INC
INX	X + 1 → X														E8	2	1																															N . . . . . Z .	INX
INY	Y + 1 → Y														C8	2	1																															N . . . . . Z .	INY
JMP	Jump to new loc					4C	3	3																															6C	6	3	7C	6	3				. . . .	JMP
JSR	Jump Subroutine					20	6	3																																								. . . .	JSR
LDA	M → A	(1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2				BD	4	3	B9	4	3										B2	5	2	N . . . . . Z .	LDA
LDX	M → X	(1)	A2	2	2	AE	4	3	A6	3	2																B6	4	2				BE	4	3													N . . . . . Z .	LDX
LDY	M → Y	(1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2				BC	4	3																N . . . . . Z .	LDY
LSR	0 → 7 0 → C	(1)				4E	6	3	46	5	2	4A	2	1										56	6	2				5E	6	3																0 . . . . . Z C	LSR
NOP	PC + 1 → PC														EA	2	1																															. . . .	NOP
ORA	A V M → A	(1)	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2				1D	4	3	19	4	3										12	5	2	N . . . . . Z .	ORA
PHA	A → M_S S - 1 → S														48	3	1																															. . . .	PHA
PHP	P → M_S S - 1 → S														08	3	1																															. . . .	PHP
PHX	X → M_S S - 1 → S														DA	3	1																															. . . .	PHX
PHY	Y → M_S S - 1 → S														5A	3	1																															. . . .	PHY
PLA	S + 1 → S M_S → A														68	4	1																															N . . . . . Z .	PLA
PLP	S + 1 → S M_S → P														28	4	1																															N V . 1 D I Z C	PLP
PLX	S + 1 → S M_S → X														FA	4	1																															N . . . . . Z .	PLX
PLY	S + 1 → S M_S → Y														7A	4	1																															N . . . . . Z .	PLY
ROL	← 7 0 ← C ←	(1)				2E	6	3	26	5	2	2A	2	1										36	6	2				3E	6	3																N . . . . . Z C	ROL
ROR	→ C → 7 0 →	(1)				6E	6	3	66	5	2	6A	2	1										76	6	2				7E	6	3																N . . . . . Z C	ROR
RTI	Return from Inter.														40	6	1																															N V . 1 D I Z C	RTI
RTS	Return from Subr.														60	6	1																															. . . .	RTS
SBC	A - M - $\bar{C}$ → A	(1,3)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2				FD	4	3	F9	4	3										F2	5	2	N V . . . . Z C	SBC
SEC	1 → C														38	2	1																															 1	SEC
SED	1 → D														F8	2	1																															. . . . 1 . . .	SED
SEI	1 → I														78	2	1																															. . . . . 1 . .	SEI
STA	A → M					8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2				9D	5	3	99	5	3										92	5	2	. . . .	STA
STX	X → M					8E	4	3	86	3	2																96	4	2																			. . . .	STX
STY	Y → M					8C	4	3	84	3	2													94	4	2																						. . . .	STY
STZ	00 → M					9C	4	3	64	3	2													74	4	2				9E	5	3																. . . .	STZ
TAX	A → X														AA	2	1																															N . . . . . Z .	TAX
TAY	A → Y														A8	2	1																															N . . . . . Z .	TAY
TRB	$\bar{A}$ ∧ M → M	(4)				1C	6	3	14	5	2																																					. . . . . . Z .	TRB
TSB	A V M → M	(4)				0C	6	3	04	5	2																																					. . . . . . Z .	TSB
TSX	S → X														BA	2	1																															N . . . . . Z .	TSX
TXA	X → A														8A	2	1																															N . . . . . Z .	TXA
TXS	X → S														9A	2	1																															. . . .	TXS
TYA	Y → A														98	2	1																															N . . . . . Z .	TYA

Notes:

1. Add 1 to "n" if page boundary is crossed.
2. Add 1 to "n" if branch occurs to same page.
   Add 2 to "n" if branch occurs to different page.
3. Add 1 to "n" if decimal mode.
4. V bit equals memory bit 6 prior to execution.
   N bit equals memory bit 7 prior to execution.

*5. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.

X	Index X	+	Add	n	No. Cycles
Y	Index Y	–	Subtract	#	No. Bytes
A	Accumulator	∧	And	M_6	Memory bit 6
M	Memory per effective address	V	Or	M_7	Memory bit 7
M_S	Memory per stack pointer	⊻	Exclusive or		

Memory Map

Appendix H explains the general rules and tables for converting numbers from one of these forms to another. For memory map diagrams, refer to Chapter 2. Figure 2-2 is an overall memory map, Figure 2-3 is a map of bank-switched memory, and Figure 2-11 is a map of the 48K memory space.

This appendix lists all important RAM and hardware locations in address order and describes them briefly. It also provides cross-references to the section of the manual where they are described further. Appendix C contains a similar list for important firmware addresses.

The tables in this appendix list addresses in either two or three forms: the hexadecimal form (preceded by a dollar sign) for use in assembly language; the decimal form for use in Applesoft BASIC; and (for numbers greater than 32767) the complementary decimal value for use in Apple Integer BASIC.

B.1 Page Zero

For Monitor zero page usage, refer to the firmware listings. For zero page use by the languages and operating systems, refer to the appropriate reference manuals.

Table B-1 lists the zero page addresses in hexadecimal and decimal form, followed by symbols denoting the firmware or system software that uses them.

- M denotes the Monitor.
- A denotes Applesoft BASIC.
- I denotes Integer BASIC.
- D denotes DOS.
- P denotes ProDOS. Locations whose contents ProDOS saves and restores afterward have a P in parentheses, indicating that ProDOS has no net effect on them.

Table B-1. Zero Page Use

Hex	Dec	Used by				
$00	0		A			
$01	1		A			
$02	2		A			
$03	3		A			
$04	4		A			
$05	5		A			
$06	6					
$07	7					
$08	8					
$09	9					
$0A	10		A			
$0B	11		A			
$0C	12		A			
$0D	13		A			
$0E	14		A			
$0F	15		A			
$10	16		A			
$11	17		A			
$12	18		A			
$13	19		A			
$14	20		A			
$15	21		A			
$16	22		A			
$17	23		A			
$18	24		A			
$19	25					
$1A	26					
$1B	27					
$1C	28					
$1D	29					
$1E	30					
$1F	31					
$20	32	M				
$21	33	M				
$22	34	M				
$23	35	M				
$24	36	M				
$25	37	M				
$26	38	M			D	
$27	39	M			D	
$28	40	M			D	
$29	41	M			D	
$2A	42	M			D	
$2B	43	M			D	
$2C	44	M			D	
$2D	45	M			D	
$2E	46	M			D	
$2F	47	M			D	
$30	48	M				
$31	49	M				
$32	50	M				
$33	51	M				
$34	52	M				
$35	53	M			D	
$36	54	M			D	
$37	55	M			D	
$38	56	M			D	
$39	57	M			D	
$3A	58	M				P
$3B	59	M				P
$3C	60	M				P
$3D	61	M				P
$3E	62	M			D	P
$3F	63	M			D	P
$40	64	M			D	(P)
$41	65	M			D	(P)
$42	66	M			D	(P)
$43	67	M			D	(P)
$44	68	M			D	(P)
$45	69	M			D	(P)
$46	70	M			D	(P)
$47	71	M			D	(P)
$48	72	M			D	(P)
$49	73	M				(P)
$4A	74			I	D	(P)
$4B	75			I	D	(P)
$4C	76			I	D	(P)
$4D	77			I	D	(P)
$4E	78	M				(P)
$4F	79	M				
$50	80	M	A			
$51	81	M	A			
$52	82	M	A			
$53	83	M	A			
$54	84	M	A			
$55	85	M	A	I		
$56	86		A	I		
$57	87		A	I		
$58	88		A	I		
$59	89		A	I		
$5A	90		A	I		
$5B	91		A	I		
$5C	92		A	I		
$5D	93		A	I		
$5E	94		A	I		
$5F	95		A	I		

Table B-1—Continued. *Zero Page Use*

Hex	Dec	Used by	Hex	Dec	Used by
$60	96	A I	$90	144	A I
$61	97	A I	$91	145	A I
$62	98	A I	$92	146	A I
$63	99	A I	$93	147	A I
$64	100	A I	$94	148	A I
$65	101	A I	$95	149	A I
$66	102	A I	$96	150	A I
$67	103	A I D	$97	151	A I
$68	104	A I D	$98	152	A I
$69	105	A I D	$99	153	A I
$6A	106	A I D	$9A	154	A I
$6B	107	A I	$9B	155	A I
$6C	108	A I	$9C	156	A I
$6D	109	A I	$9D	157	A I
$6E	110	A I	$9E	158	A I
$6F	111	A I D	$9F	159	A I
$70	112	A I D	$A0	160	A I
$71	113	A I	$A1	161	A I
$72	114	A I	$A2	162	A I
$73	115	A I	$A3	163	A I
$74	116	A I	$A4	164	A I
$75	117	A I	$A5	165	A I
$76	118	A I	$A6	166	A I
$77	119	A I	$A7	167	A I
$78	120	A I	$A8	168	A I
$79	121	A I	$A9	169	A I
$7A	122	A I	$AA	170	A I
$7B	123	A I	$AB	171	A I
$7C	124	A I	$AC	172	A I
$7D	125	A I	$AD	173	A I
$7E	126	A I	$AE	174	A I
$7F	127	A I	$AF	175	A I D
$80	128	A I	$B0	176	A I D
$81	129	A I	$B1	177	A I
$82	130	A I	$B2	178	A I
$83	131	A I	$B3	179	A I
$84	132	A I	$B4	180	A I
$85	133	A I	$B5	181	A I
$86	134	A I	$B6	182	A I
$87	135	A I	$B7	183	A I
$88	136	A I	$B8	184	A I
$89	137	A I	$B9	185	A I
$8A	138	A I	$BA	186	A I
$8B	139	A I	$BB	187	A I
$8C	140	A I	$BC	188	A I
$8D	141	A I	$BD	189	A I
$8E	142	A I	$BE	190	A I
$8F	143	A I	$BF	191	A I

Table B-1—Continued. Zero Page Use

Hex	Dec	Used by			Hex	Dec	Used by
$C0	192	A	I		$E0	224	A
$C1	193	A	I		$E1	225	A
$C2	194	A	I		$E2	226	A
$C3	195	A	I		$E3	227	
$C4	196	A	I		$E4	228	A
$C5	197	A	I		$E5	229	A
$C6	198	A	I		$E6	230	A
$C7	199	A	I		$E7	231	A
$C8	200	A	I		$E8	232	A
$C9	201	A	I		$E9	233	A
$CA	202	A	I	D	$EA	234	A
$CB	203	A	I	D	$EB	235	
$CC	204	A	I	D	$EC	236	
$CD	205	A	I	D	$ED	237	
$CE	206		I		$EE	238	
$CF	207		I		$EF	239	
$D0	208	A	I		$F0	240	A
$D1	209	A	I		$F1	241	A
$D2	210	A	I		$F2	242	A
$D3	211	A	I		$F3	243	A
$D4	212	A	I		$F4	244	A
$D5	213	A	I		$F5	245	A
$D6	214		I		$F6	246	A
$D7	215		I		$F7	247	A
$D8	216	A	I	D	$F8	248	A
$D9	217	A	I		$F9	249	
$DA	218	A	I		$FA	250	
$DB	219	A	I		$FB	251	
$DC	220	A	I		$FC	252	
$DD	221	A	I		$FD	253	
$DE	222	A	I		$FE	254	
$DF	223	A	I		$FF	255	

B.2 Page Three

Most of page 3 is available for small machine-language programs or any other use your program might put it to. The built-in Monitor uses the top sixteen addresses of page 3, as shown in Table B-2; the XFER routine (section 2.5.3) uses locations $3ED and $3EE. If you are using DOS or ProDOS, it also uses the 32 locations $3D0 through $3EF.

Table B-2. *Page 3 Use*

Hex	Dec	Section	Use
$3F0	1008	2.6.4	Address of BRK request handler (normally $59, $FA)
$3F1	1009		
$3F2	1010	2.6.4 & 10.1	Reset vector
$3F3	1011		
$3F4	1012	2.6.4	Power-up byte (see text)
$3F5	1013		Jump instruction to Applesoft &-command handler (initially $4C, $58, $FF)
$3F6	1014		
$3F7	1015		
$3F8	1016	10.6.4	Jump instruction to user CONTROL-Y command handler
$3F9	1017		
$3FA	1018		
$3FB	1019		Jump instruction to NMI interrupt handler (not used by Apple IIc)
$3FC	1020		
$3FD	1021		
$3FE	1022	2.6.4	Address of user IRQ interrupt handler
$3FF	1023		

B.3 Screen Holes

One result of the way the Apple IIc hardware maps display memory on the screen is that groups of eight memory addresses are left over in sixteen areas of the text and low-resolution display pages—eight areas in main RAM and eight in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4, with cross-references to the section numbers where they are described.

Table B-3. *Main Memory Screen Hole Allocations*

Hex	Dec	Section	Description
$478	1144	9.1.5	Mouse port: low byte of clamping minimum
$479	1145	7.5	Reserved for serial port 1
$47A	1146	8.5	Reserved for serial port 2
$47B	1147		Reserved
$47C	1148	9.1.5	Low byte of X coordinate
$47D	1149		Reserved for mouse port
$47E	1150		Reserved
$47F	1151		Reserved
$4F8	1272	9.1.5	Mouse port: low byte of clamping maximum
$4F9	1273	7.5,E.6.3	Reserved for serial port 1
$4FA	1274	8.5,E.6.2	Reserved for serial port 2
$4FB	1275		Reserved
$4FC	1276	9.1.5	Low byte of Y coordinate
$4FD	1277		Reserved for mouse port
$4FE	1278		Reserved
$4FF	1279	E.6.4	Reserved
$578	1400	9.1.5	Mouse port: high byte of clamping minimum
$579	1401	7.5	Port 1 printer width (1-255; 0 = unlimited)
$57A	1402	8.5	Port 2 line length (1-255; 0 = unlimited)
$57B	1403		Cursor horizontal position (80-column display)
$57C	1404	9.1.5	High byte of X coordinate
$57D	1405		Reserved for mouse port
$57E	1406		Reserved
$57F	1407	E.6.4	Reserved
$5F8	1528	9.1.5	Mouse port: high byte of clamping maximum
$5F9	1529	7.5,E.6.3	Port 1 temporary storage location
$5FA	1530	8.5,E.6.2	Port 2 temporary storage location
$5FB	1531		Reserved
$5FC	1532	9.1.5	High byte of Y coordinate
$5FD	1533		Reserved for mouse port
$5FE	1534		Reserved
$5FF	1535	E.6.2	Reserved

Table B-3—Continued. Main Memory Screen Hole Allocations

Hex	Dec	Section	Description
$678	1656		Reserved
$679	1657	7.5	Indicates when port 1 firmware is parsing a command
$67A	1658	8.5	Indicates when port 2 firmware is parsing a command
$67B	1659		Reserved
$67C	1660	9.1.5	Mouse port: reserved
$67D	1661		Reserved for mouse port
$67E	1662		Reserved
$67F	1663	E.6.4	Reserved
$6F8	1784		Reserved
$6F9	1785	7.5	Current port 1 command character
$6FA	1786	8.5	Current port 2 command character
$6FB	1787		Reserved
$6FC	1788	9.1.5	Mouse port: reserved
$6FD	1789		Reserved for mouse port
$6FE	1790		Reserved
$6FF	1791	E.6.2	Reserved
$778	1912		DEVNO: $n0 = current active port number x 16
$779	1913	7.5	Port 1 flags for echo and auto line feed
$77A	1914	8.5	Port 2 flags for echo and auto line feed
$77B	1915		Reserved
$77C	1916	9.1.5,E.6.1	Mouse port status byte
$77D	1917		Reserved for mouse port
$77E	1918		Reserved
$77F	1919		Reserved
$7F8	2040		MSLOT: owner of $C800-$CFFF ($C3, video)
$7F9	2041	7.5	Port 1 current printer column
$7FA	2042	8.5	Port 2 current line position
$7FB	2043		Reserved
$7FC	2044	9.1.5	Mouse port mode byte
$7FD	2045		Reserved for mouse port
$7FE	2046		Reserved
$7FF	2047		Reserved

Table B-4. Auxiliary Memory Screen Hole Allocations

Hex	Dec	Section	Description
$478	1144	7.5	Initial port 1 ACIA Control Register values ($9E)
$479	1145	7.5	Initial port 1 ACIA Command Register values ($0B)
$47A	1146	7.5	Initial port 1 characteristics flags ($40)
$47B	1147	7.5	Initial port 1 printer width ($50)
$47C	1148	8.5	Initial port 2 ACIA Control Register values ($16)
$47D	1149	8.5	Initial port 2 ACIA Command Register values ($0B)
$47E	1150	8.5	Initial port 2 characteristics flags ($01)
$47F	1151	8.5	Initial port 2 line length ($00)
$4F8 through $4FF	1272 1279		Reserved
$578 through $57F	1400 1407		Reserved
$5F8 through $5FF	1528 1535		Reserved
$678 through $67F	1656 1663		Reserved
$6F8 through $6FF	1784 1791		Reserved
$778 through $77F	1912 1919		Reserved
$7F8 through $7FF	2040 2047		Reserved

B.4 The Hardware Page

Tables B-5 through B-9 list all the hardware locations available for use in the Apple IIc. These tables have a column at the left that is not present in other tables. This column, labeled RW, indicates the action to take at a particular location.

- R means read.
- RR means read twice in succession.
- R7 means read the byte and then check bit 7; in the use column, "see if..." refers to the condition represented by bit 7 = 1, unless otherwise specified. Bit 7 has a value of $80, so if the contents of the location are greater than or equal to $80, the bit is on.

  Another way to test bit 7 (the sign bit) is with a BIT instruction, followed by BPL (bit 7 was 0) or BMI (bit 7 was 1).
- R/W means to either read or write. For writing, the value is unimportant.
- W means to write only. The value is unimportant.
- N means not to read or write, because the location is reserved.

An address of the form $C00x means the sixteen locations from $C000 through $C00F. Labels, when they are shown, are simply memory aids. Some of them correspond to the labels at those addresses in the firmware, others do not. Your program will have to assign a label for it anyway.

Table B-5. Addresses $C000 Through $C03F

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	$C00x			KSTRB	4.1	Read keyboard data (bits 0-6) and strobe (bit 7)
W	$C000	49152	-16384	80STORE	5.6†	Off: PAGE2 switches Page 1 and 2
W	$C001	49153	-16383	80STORE	5.6†	On: PAGE2 switches Page 1 and 1X
W	$C002	49154	-16382	RAMRD	2.5.2	Off: read main 48K RAM
W	$C003	49155	-16381	RAMRD	2.5.2	On: read auxiliary 48K RAM
W	$C004	49156	-16380	RAMWRT	2.5.2	Off: write in main 48K RAM
W	$C005	49157	-16379	RAMWRT	2.5.2	On: write in auxiliary 48K RAM
W	$C006	49158	-16378			Reserved
W	$C007	49159	-16377			Reserved
W	$C008	49160	-16376	ALTZP	2.4.2	Off: use main P0, P1, bank-switched RAM
W	$C009	49161	-16375	ALTZP	2.4.2	On: use auxiliary P0, P1, bank-switched RAM
W	$C00A	49162	-16374			Reserved
W	$C00B	49163	-16373			Reserved
W	$C00C	49164	-16372	80COL	5.6	Off: 40-column display
W	$C00D	49165	-16371	80COL	5.6	On: 80-column display
W	$C00E	49166	-16270	ALTCHAR	5.6	Off: display primary character set
W	$C00F	49167	-16369	ALTCHAR	5.6	On: display alternate character set
W	$C01x				4.1	Clear keyboard strobe ($C00x bit 7)
R7	$C010	49168	-16368	AKD	4.1	See if any key now down; clear strobe
R7	$C011	49169	-16367	RDBNK2	2.4.2	See if using $D000 bank 2 (or 1)
R7	$C012	49170	-16366	RDLCRAM	2.4.2	See if reading RAM (or ROM).
R7	$C013	49171	-16365	RDRAMRD	2.5.2	See if reading auxiliary 48K RAM (or main)
R7	$C014	49172	-16364	RDRAMWRT	2.5.2	See if writing auxiliary 48K RAM (or main)
R	$C015	49173	-16363	RSTXINT	9.1.3	Reset mouse X0 interrupt.
R7	$C016	49174	-16362	RDALTZP	2.4.2	See if auxiliary P0, P1 and bank-switched RAM
R	$C017	49175	-16361	RSTYINT	9.1.3	Reset mouse Y interrupt
R7	$C018	49176	-16360	RD80STORE	5.6†	See if 80STORE on (or off)
R7	$C019	49177	-16359	RSTVBL	9.1.3	See if VBLINT off (1); reset it
R7	$C01A	49178	-16358	RDTEXT	5.6	See if text (or graphics)
R7	$C01B	49179	-16357	RDMIX	5.6	See if mixed mode switch on
R7	$C01C	49180	-16356	RDPAGE2	5.6†	See if page 2/1X selected (or 1)
R7	$C01D	49181	-16355	RDHIRES	5.6†	See if high-resolution switch on
R7	$C01E	49182	-16354	RDALTCHAR	5.6	See if alternate character set (or primary)
R7	$C01F	49183	-16353	RD80COL	5.6	See if 80-column hardware on
N	$C020	49184	-16352			Reserved (read and write)
	through					
N	$C02F	49199	-16337			
W	$C030	49200	-16336			Reserved
R	$C030	49200	-16336		4.2.1	Toggle speaker
N	$C031	49201	-16335			Reserved (read and write)
	through					
N	$C03F	49215	-16321			

† Also section 2.5.4

Table B-6. *Addresses $C040 Through $C05F*

RW	Hex	Dec	Neg Dec	Label	Section	Use
R7	$C040	49216	-16320	RDXYMSK	9.1.3	See if X0/Y0 mask set
R7	$C041	49217	-16319	RDVBLMSK	9.1.3	See if VBL mask set
R7	$C042	49218	-16318	RDX0EDGE	9.1.3	See if interrupt on falling X0 edge
R7	$C043	49219	-16317	RDY0EDGE	9.1.3	See if interrupt on falling Y0 edge
N	$C044	49220	-16316			Reserved
N	$C045	49221	-16315			Reserved
N	$C046	49222	-16314			Reserved
N	$C047	49223	-16313			Reserved
R	$C048	49224	-16312	RSTXY	9.1.3	Reset X0/Y0 interrupt flags
N	$C049	49225	-16311			Reserved
N	$C04A	49226	-16310			Reserved
N	$C04B	49227	-16309			Reserved
N	$C04C	49228	-16308			Reserved
N	$C04D	49229	-16307			Reserved
N	$C04E	49230	-16306			Reserved
N	$C04F	49231	-16305			Reserved
R/W	$C050	49232	-16304	TEXT	5.6	Off: graphics display
R/W	$C051	49233	-16303	TEXT	5.6	On: text display
R/W	$C052	49234	-16302	MIXED	5.6	Off: text or graphics only
R/W	$C053	49235	-16301	MIXED	5.6	On: combination text and graphics
R/W	$C054	49236	-16300	PAGE2	5.6†	Off: use page 1
R/W	$C055	49237	-16299	PAGE2	5.6†	On: display page 2 (80STORE off); store to page 1X (80STORE on)
R/W	$C056	49238	-16298	HIRES	5.6†	Off: low-resolution
R/W	$C057	49239	-16297	HIRES	5.6†	On: high-resolution; double if 80COL and DHIRES on
N	$C058	49240	-16296			Reserved if IOUDIS on ($C07E bit 7=1)
R/W				DISXY	9.1.3	Disable (mask) mouse X0/Y0 interrupts
N	$C059	49241	-16295			Reserved if IOUDIS on
R/W				ENBXY	9.1.3	Enable (allow) mouse X0/Y0 interrupts
N	$C05A	49242	-16294			Reserved if IOUDIS on
R/W				DISVBL	9.1.3	Disable (mask) VBL interrupts
N	$C05B	49243	-16293			Reserved if IOUDIS on
R/W				ENVBL	9.1.3	Enable (allow) VBL interrupts
N	$C05C	49244	-16292			Reserved if IOUDIS on
R/W				X0EDGE	9.1.3	Interrupt on rising edge of X0
N	$C05D	49245	-16291			Reserved if IOUDIS on
R/W				X0EDGE	9.1.3	Interrupt on falling edge of X0
R/W	$C05E	49246	-16290	DHIRES	5.6	If IOUDIS on: set double-high-resolution
R/W				Y0EDGE	9.1.3	If IOUDIS off: interrupt on rising Y0
R/W	$C05F	49247	-16289	DHIRES	5.6	If IOUDIS on: clear double-high-resolution
R/W				Y0EDGE	9.1.3	If IOUDIS off: interrupt on falling Y0

† Also section 2.5.4.

Table B-7. Addresses $C060 Through $C07F

RW	Hex	Dec	Neg Dec	Label	Section	Use
W	$C06x					Reserved (write)
R7	$C060	49248	-16288	RD80SW	4.1	See if 80/40 switch down (= 40)
R7	$C061	49249	-16287	RDBTN0	9.1.3†	See if switch 0 or (○) pressed
R7	$C062	49250	-16286	RDBTN1	9.2†	See if switch 1 or (●) pressed
R7	$C063	49251	-16285	RD63	9.1,9.2	See if mouse button not pressed
R7	$C064	49252	-16284	PDL0	9.2	See if hand control button 0 pressed
R7	$C065	49253	-16283	PDL1	9.2	See if hand control button 1 pressed
R7	$C066	49254	-16282	MOUX1	9.1.3	See if mouse X1 (direction) is high
R7	$C067	49255	-16281	MOUY1	9.1.3	See if mouse Y1 (direction) is high
N	$C068	49256	-16280			
	through					Reserved (write and read)
N	$C06F	49263	-16273			
R/W	$C07x					Trigger paddle timer; reset VBLINT; however, some $C07x are reserved
R/W	$C070	49264	-16272	PTRIG	9.2	Designated trigger or reset location
N	$C071	49265	-16271			
	through					Reserved
N	$C07D	49277	-16259			
R7	$C07E	49278	-16258	RDIOUDIS		See if IOUDIS on; trigger paddle timer; reset VBLINT
W				IOUDIS	5.6,9.1.3	On: enable access to DHIRES switch; disable $C058-$C05F IOU access
R7	$C07F	49279	-16257	RDDHIRES	5.6,9.1.3	See if DHIRES on
W				IOUDIS	5.6	Off: disable access to DHIRES switch; enable $C058-$C05F IOU access

† Also section 4.1.

Table B-8. Addresses $C080 Through $C0AF

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	$C080	49280	-16256		2.4.2	Read RAM; no write; use $D000 bank 2
RR	$C081	49281	-16255		2.4.2	Read ROM, write RAM; use $D000 bank 2
R	$C082	49282	-16254		2.4.2	Read ROM; no write; use $D000 bank 2
RR	$C083	49283	-16253		2.4.2	Read and write RAM; use $D000 bank 2
N	$C084	49284	-16252			Reserved
N	$C085	49285	-16251			Reserved
N	$C086	49286	-16250			Reserved
N	$C087	49287	-16249			Reserved
R	$C088	49288	-16248		2.4.2	Read RAM; no write; use $D000 bank 1
RR	$C089	49289	-16247		2.4.2	Read ROM, write RAM; use $D000 bank 1
R	$C08A	49290	-16246		2.4.2	Read ROM; no write; use $D000 bank 1
RR	$C08B	49291	-16245		2.4.2	Read and write RAM; use $D000 bank 1
N	$C08C	49292	-16244			Reserved
N	$C08D	49293	-16243			Reserved
N	$C08E	49294	-16242			Reserved
N	$C08F	49295	-16241			Reserved
N	$C090	49296	-16240			Reserved
	through					
N	$C097	49303	-16233			
R/W	$C098	49304	-16232		7.3, 11.11	Port 1 ACIA Transmit/receive register
R/W	$C099	49305	-16231		7.3, 11.11	Port 1 ACIA Status register
R/W	$C09A	49306	-16230		7.3, 11.11, Appendix E	Port 1 ACIA Command register
R/W	$C09B	49307	-16229		7.3, 11.11	Port 1 ACIA Control register
N	$C09C	49308	-16228			Reserved
	through					
N	$C09F	49311	-16225			
N	$C0A0	49312	-16224			Reserved
	through					
N	$C0A7	49319	-16217			
R/W	$C0A8	49320	-16216		8.3, 11.11	Port 2 ACIA Transmit/receive register
R/W	$C0A9	49321	-16215		8.3, 11.11	Port 2 ACIA Status register
R/W	$C0AA	49322	-16214		8.3, 11.11, Appendix E	Port 2 ACIA Command register
R/W	$C0AB	49323	-16213		8.3, 11.11	Port 2 ACIA Control register
N	$C0AC	49324	-16212			Reserved
	through					
N	$C0AF	49327	-16209			

Table B-9. Addresses $C0B0 Through $C0FF

RW	Hex	Dec	Neg Dec	Label	Section	Use
N	$C0B0	49328	-16208			
	through					Reserved
N	$C0BF	49343	-16193			
N	$C0C0	49344	-16192			
	through					Reserved
N	$C0CF	49359	-16177			
N	$C0D0	49360	-16176			
	through					Reserved
N	$C0DF	49375	-16161			
N	$C0E0	49376	-16160			
	through					Reserved
N	$C0EF	49391	-16145			
N	$C0F0	49392	-16144			
	through					Reserved
N	$C0FF	49407	-16129			

Important Firmware Locations

appendix c

This appendix lists all significant firmware addresses: entry points, locations containing the addresses of entry points, and locations where machine and device identification bytes reside.

▲

Warning
The Monitor firmware entry points are the only published *entry points in the sense that they are the only ones that will remain in the same locations in future Apple II series computers.*

The firmware protocol identification bytes and offsets will work with other Apple II series computers only if used as directed (section 3.4.2).

C.1 The Tables

Appendix H contains tables and examples of the derivation of each form of address from either of the other forms.

This appendix supplements the chapter text by specifying three forms of each address: hexadecimal, decimal, and complementary (negative) decimal.

In these tables, some of the addresses are followed by a label of the location. These labels are listed only to assist you in finding the named location in the firmware listings, or in remembering the function found at the address. The Apple IIc contains no global label table: your program must assign its own labels to the addresses as required.

There are several types of information at these firmware addresses: actual entry points (labeled *entry*), the low-order byte of an entry point (labeled *offset*), a device or machine identification byte (labeled *ident),* indicators (labeled *indic)* specifying whether there are optional routines, vector addresses (labeled *vector),* and an RTS instruction location.

The column labeled *Section* contains the number of the section that describes the item. If there is no description except in a table in this appendix, a section number is not listed.

Each input/output port has an associated protocol table, as shown in Tables C-1 through C-4. Many of the bytes (labeled *offset*) in the protocol tables are the low-order bytes of addresses of I/O routines for the ports; the high-order byte of these addresses must be $Cn (where *n* is the port number). This structure is explained in Chapter 3. Although your program must perform some extra processing to use these tables, the benefit is simplified compatible port and slot I/O for all Apple II series machines.

C.2 Port Addresses

Table C-1. *Serial Port 1 Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C100	49408	-16128		entry	3.1.1	Main port 1 entry point
$C105	49413	-16123		ident	3.4.2	ID byte ($38)
$C107	49415	-16121		ident	3.4.2	ID byte ($18)
$C10B	49419	-16117		ident	3.4.2	Firmware card signature ($01)
$C10C	49420	-16116		ident	3.4.2	Super Serial Card ID ($31)
$C10D	49421	-16115		offset	7.4	Low-order PINIT address
$C10E	49422	-16114		offset	7.4	Low-order PREAD address
$C10F	49423	-16113		offset	7.4	Low-order PWRITE address
$C110	49424	-16112		offset	7.4	Low-order PSTATUS address
$C111	49425	-16111		indic	3.4.2	Non-zero: no optional routines

Table C-2. *Serial Port 2 Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C200	49664	-15872		entry	3.1.1	Main port 2 entry point
$C205	49669	-15867		ident	3.4.2	ID byte ($38)
$C207	49671	-15865		ident	3.4.2	ID byte ($18)
$C20B	49675	-15861		ident	3.4.2	Firmware card ID ($01)
$C20C	49676	-15860		ident	3.4.2	Super Serial Card ID ($31)
$C20D	49677	-15859		offset	8.4	Low-order PINIT address
$C20E	49678	-15858		offset	8.4	Low-order PREAD address
$C20F	49679	-15857		offset	8.4	Low-order PWRITE address
$C210	49680	-15856		offset	8.4	Low-order PSTATUS address
$C211	49681	-15855		indic	3.4.2	Non-zero: no optional routines

Table C-3. *Video Firmware Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C300	49920	-15616		entry	3.1.1	Main video entry point (output only)
$C305	49925	-15611	C3KEYIN	ident	3.4.2	ID byte ($38)
$C307	49927	-15609	C3COUT1	ident	3.4.2	ID byte ($18)
$C30B	49931	-15605		ident	3.4.2	Firmware card signature ($01)
$C30C	49932	-15604		ident	3.4.2	80-column card ID ($88)
$C30D	49933	-15603		offset	5.9	Low-order PINIT address
$C30E	49934	-15602		offset	5.9	Low-order PREAD address
$C30F	49935	-15601		offset	5.9	Low-order PWRITE address
$C310	49936	-15600		offset	5.9	Low-order PSTATUS address
$C311	49937	-15599	MOVEAUX	entry	2.5.3	Routine for main/auxiliary control swapping (Also called AUXMOVE)

Table C-4. *Mouse Port Addresses*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C400	50176	-15360		entry		Main mouse entry point
$C405	50181	-15355		ident	3.4.2	ID byte ($38)
$C407	50183	-15353		ident	3.4.2	ID byte ($18)
$C40B	50187	-15349		ident	3.4.2	Firmware card signature ($01)
$C40C	50188	-15348		type	3.4.2	X-Y pointing device ID ($20)
$C40D	50189	-15347		offset	9.1.4	Low-order PINIT address
$C40E	50190	-15346		offset	9.1.4	Low-order PREAD address
$C40F	50191	-15345		offset	9.1.4	Low-order PWRITE address
$C410	50192	-15344		offset	9.1.4	Low-order PSTATUS address
$C411	50193	-15343		indic	3.4.2	Optional routines follow ($00)
$C412	50194	-15342	SETMOUSE	offset	9.1.4	Low-order SETMOUSE address
$C413	50195	-15341	SERVEMOUSE	offset	9.1.4	Low-order SERVEMOUSE address
$C414	50196	-15340	READMOUSE	offset	9.1.4	Low-order READMOUSE address
$C415	50197	-15339	CLEARMOUSE	offset	9.1.4	Low-order CLEARMOUSE address
$C416	50198	-15338	POSMOUSE	offset	9.1.4	Low-order POSMOUSE address
$C417	50199	-15337	CLAMPMOUSE	offset	9.1.4	Low-order CLAMPMOUSE address
$C418	50200	-15336	HOMEMOUSE	offset	9.1.4	Low-order HOMEMOUSE address
$C419	50201	-15335	INITMOUSE	offset	9.1.4	Low-order INITMOUSE address

C.3 Other Video and I/O Firmware Addresses

Miscellaneous firmware addresses are listed in Table C-5.

Table C-5. *Apple IIc Enhanced Video and Miscellaneous Firmware*

Hex	Dec	Neg Dec	Label	Type	Section	Description
$C600	50688	-14848		entry	6.1	Disk drive firmware entry point
$C700	50944	-14592		entry	6.2	External disk startup routine
$C803	51203	-14333	NEWIRQ	entry	E.1	IRQ handling routine

C.4 Applesoft BASIC Interpreter Addresses

The addresses of Applesoft BASIC entry points are listed in the *Applesoft BASIC Programmer's Reference Manual.* The Applesoft interpreter occupies ROM addresses from $D000 through $F7FF.

C.5 Monitor Addresses

Table C-6 lists the Monitor entry points, machine identifier bytes, interrupt vectors, and the address of a known RTS instruction.

Table C-6. Apple IIc Monitor Entry Points and Vectors

Hex	Dec	Neg Dec	Label	Type	Section	Description
$F800	63488	-2048	PLOT	entry	5.8	Plots a low-resolution block
$F819	63513	-2023	HLINE	entry	5.8	Draws low-resolution horizontal line
$F828	63528	-2008	VLINE	entry	5.8	Draws low-resolution vertical line
$F832	63538	-1998	CLRSCR	entry	5.8	Clears low-resolution screen
$F836	63542	-1994	CLRTOP	entry	5.8	Clears top 40 low-resolution lines
$F864	63588	-1948	SETCOL	entry	5.8	Sets low-resolution color (Table 5-4)
$F871	63601	-1935	SCRN	entry	5.8	Reads color of low-resolution block
$F941	63809	-1727	PRNTAX	entry	5.8	Displays (A) and (X) in hex
$F94A	63818	-1718	PRBL2	entry	5.8	Sends (X) blanks to output
$FA47	63845	-1691	NEWBRK	entry	E.2	Apple IIc break handler
$FA62	64098	-1438	RESET	entry	2.6	Hardware reset routine
$FB1E	64286	-1250	PREAD	entry	9.2	Reads hand control position
$FB6F	64367	-1169	SETPWRC	entry	2.6.4	Routine to create power-up byte
$FBB3	64435	-1101		ident	F.1.2	Machine identification byte
$FBC0	64448	-1088		ident	F.1.2	Machine identification byte
$FBDD	64477	-1059	BELL1	entry	4.2.2	Sends 1 kHz beep to speaker
$FC42	64578	-958	CLREOP	entry	5.8	Clears from cursor to bottom
$FC58	64600	-936	HOME	entry	5.8	Clears; cursor to upper left
$FC9C	64668	-868	CLREOL	entry	5.8	Clears from cursor to end of line
$FC9E	64670	-866	CLEOLZ	entry	5.8	Clears from BASL to end of line
$FCA8	64680	-856	WAIT	entry		Delays for time specified by (A)
$FD0C	64780	-756	RDKEY	entry	3.2.1	Displays cursor, jumps to (KSW)
$FD1B	64795	-741	KEYIN	entry	3.2.2	Waits for keypress, reads key
$FD35	64821	-715	RDCHAR	entry	4.1.2	Gets input, interprets ESC codes
$FD67	64871	-665	GETLNZ	entry	4.1.2	Sends CR to output, goes to GETLN
$FD6A	64874	-662	GETLN	entry	3.2.3	Displays prompt, gets input line
$FD6F	64879	-657	GETLN1	entry	4.1.2	No prompt; gets input line
$FD8B	64907	-629	CROUT1	entry	5.8	Clears to end of line, calls CROUT
$FD8E	64910	-626	CROUT	entry	5.8	Sends CR to output
$FDDA	64986	-550	PRBYTE	entry	5.8	Sends (A) to output
$FDE3	64995	-541	PRHEX	entry	5.8	Displays low nibble of (A) in hex
$FDED	65005	-531	COUT	entry	3.3.1	Jumps to (CSW)
$FDF0	65008	-528	COUT1	entry	3.3.2	Displays (A), advances cursor
$FE2C	65068	-468	MOVE	entry		Copies (memory) elsewhere
$FE36	65078	-458	VERIFY	entry		Compares two blocks of memory
$FF2D	65325	-211	PRERR	entry	5.8	Sends ERR to output; beeps
$FF3A	65338	-198	BELL	entry	4.2.2	Sends CONTROL-G to output
$FF3F	65343	-193	IOREST	entry		Loads ($45-$49) into registers
$FF4A	65354	-182	IOSAVE	entry		Stores (A,X,Y,P,S) at $45-$49
$FF58	65368	-168	IORTS	RTS		Location of known RTS instruction
$FF69	65385	-151	(Monitor)	entry	10.1	Standard Monitor entry point
$FFFA	65530	-6		vector		Low-order NMI vector (unused)
$FFFB	65531	-5		vector		High-order NMI vector (unused)
$FFFC	65532	-4		vector		Low-order RESET vector ($62)
$FFFD	65533	-3		vector		High-order RESET vector ($FA)
$FFFE	65534	-2	IRQVECT	vector		Low-order IRQ vector ($03)
$FFFF	65535	-1		vector		High-order IRQ vector ($CB)

Operating Systems and Languages

This appendix is an overview of the characteristics of operating systems and languages when run on the Apple IIc. It is not intended to be a full account. For more information, refer to the manuals that are provided with each product.

D.1 Operating Systems

This section discusses the operating systems that the Apple IIc does and does not work with.

D.1.1 ProDOS

ProDOS is the preferred disk operating system for the Apple IIc. It supports startup from the external disk drive, interrupts, and all other hardware and firmware features of the Apple IIc.

D.1.2 DOS

The Apple IIc works with DOS 3.3. Its disk drive support hardware and firmware can also access DOS 3.2 disks by using the *BASICS* disk. However, neither version of DOS takes full advantage of the features of the Apple IIc. DOS support is provided only for the sake of Apple II series compatibility.

D.1.3 Pascal Operating System

Version 1.2 of the Pascal Operating System uses the 80/40 switch and the interrupt features of the Apple IIc, while remaining compatible with the other Apple II series computers.

While the Apple IIc works with Pascal 1.1, this version of the Pascal Operating System does not use the 80/40 switch or handle interrupts.

The Apple IIc does not work with Pascal 1.0, because the input/output firmware entry points are rigidly defined (rather than being accessed via a table), and the firmware does not support these entry points.

D.1.4 CP/M

CP/M, and any other operating system that requires an interface card, will not work on the Apple IIc.

D.2 Languages

For further information about these languages, refer to the manuals that came with them.

This section discusses special techniques to use, and characteristics to be aware of, when using Apple programming languages with the Apple IIc. It is also a guide to using this reference manual with these languages.

D.2.1 Applesoft BASIC

Use the appendixes to make or find decimal conversions. Appendix H has tables and examples to help you convert numbers between hexadecimal, decimal, and negative (complementary) decimal. All the addresses listed in Appendixes B and C—screen holes, hardware addresses, firmware entry points, and so on—are given in all three numeric forms.

The focus of the chapters in this manual is assembly language, and so most addresses and values are given in hexadecimal notation.

Use a PEEK in BASIC (instead of LDA in assembly language) to read a location, and a POKE (instead of STA) to write to a location. If you read a hardware address from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if a soft switch is on, its value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

D.2.2 Integer BASIC

Unless you load a version of DOS into your Apple IIc, you will not have Integer BASIC available inside the machine. ProDOS does not support Integer BASIC.

D.2.3 Pascal Language

The Pascal language works on the Apple IIc under versions 1.1 and 1.2 of the Pascal Operating System. However, for best performance, use Pascal version 1.2.

D.2.4 FORTRAN

FORTRAN works under version 1.1 of the Pascal Operating System which, as explained in section D.1.3, does not detect or use certain Apple IIc features, such as the 80/40 switch or auxiliary memory. Therefore, FORTRAN does not take advantage of these features either.

Interrupts

This appendix presents a unified account of the sources of interrupts on the Apple IIc, how the firmware handles the interrupts, and how to use interrupt-driven features directly in those rare cases when the firmware cannot meet your needs.

Warning
If you use interrupt hardware directly, rather than using the built-in interrupt-handling firmware, compatibility with possible future Apple II series computers or revisions cannot be guaranteed.

E.1 Introduction

This section orients you to interrupts and their effects on the Apple IIc hardware.

E.1.1 What Is an Interrupt?

On a computer, an interrupt is a signal that abruptly causes the computer to stop what it is currently doing and immediately attend to an important time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This is necessary because unless the mouse is read shortly after it moves, the signal indicating its direction is lost.

When an interrupt occurs, control passes to an interrupt handler, which must record the exact state of the computer at the moment of the interrupt, determine the source of the interrupt, and take appropriate action. It is important that the computer preserve a *snapshot* of its state when interrupted, so that when it continues later with what it had been doing, those conditions can be restored.

E.1.2 Interrupts on Apple II Computers

Interrupts have not always been fully supported on the Apple II. All versions of Apple's DOS, as well as the Monitor program, rely on the integrity of location $45, which the built-in interrupt handler has always destroyed by saving the accumulator in it. Most versions of Pascal simply do not work with interrupts enabled.

The Apple IIc built-in interrupt handler now saves the accumulator on the stack instead of in location $45. Thus both DOS and the Monitor work with interrupts on the Apple IIc.

If, however, you want software that uses interrupts to work on the Apple IIe and the Apple II Plus, you must use either ProDOS, Apple's new enhanced disk operating system, or Pascal 1.2. Both operating systems have full interrupt support built in.

Interrupts are effective only if they are enabled most of the time. Interrupts that occur while interrupts are disabled cannot be detected. Due to the critical timing of disk read and write operations, Pascal, DOS, and ProDOS turn off interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all sources of interrupts discussed below are turned off.

On the Apple IIe only, interrupts are periodically turned off while 80-column screen operations are being performed. This is most noticeable while the screen is scrolling. Also, most peripheral cards used in the Apple IIe disable interrupts while reading and writing.

E.1.3 Interrupt Handling on the 65C02

From the point of view of the 65C02, there are three possible causes of interrupts.

1. If 65C02 interrupts are not masked (that is, the CLI instruction has been used), the IRQ line on the microprocessor can be pulled low. This is the standard technique by which a device indicates that it needs immediate attention.
2. The processor executes a break (BRK, opcode $00) instruction.
3. A non-maskable interrupt (NMI) occurs. Because the NMI line in the Apple IIc's 65C02 is not used, this never happens.

Options 1 and 2 cause the 65C02 to save the current program counter and status byte on the stack and then jump to the routine whose address is stored in $FFFE and $FFFF. The sequence performed by the 65C02 is:

1. If IRQ, finish executing the current instruction. (If BRK, current instruction is already finished.)
2. Push high byte of program counter onto stack.
3. Push low byte of program counter onto stack.
4. Push program status byte onto stack.
5. Jump to address stored in $FFFE, $FFFF, that is, JMP ($FFFE).

The different sources of interrupt signals are discussed below.

E.1.4 The Interrupt Vector at $FFFE

In the Apple IIc computer there are three separate regions of memory that contain address $FFFE: the built-in ROM, the bank-switched memory in main RAM, and the bank-switched memory in auxiliary RAM. The vector at $FFFE in the ROM points to Apple IIc's built-in interrupt handling routine. Due to the complexity of interrupts in the Apple IIc, it is recommended that you use it rather than writing your own interrupt handling routine.

When you initialize the mouse or serial communication firmware, copies of the ROM's interrupt vector are placed in the interrupt vector addresses in both main and auxiliary bank-switched memory. If you plan to use interrupts and the bank-switched memory without the mouse or communication firmware, you must copy the ROM's interrupt vector yourself.

E.2 The Built-in Interrupt Handler

The built-in interrupt handler is responsible for determining whether a BRK or an IRQ interrupt occurred. If it was an IRQ interrupt, it decides whether the interrupt should be handled internally, handled by the user, or simply ignored.

The built-in interrupt handling routine records the current memory configuration, then sets up its own standard memory configuration so that a user's interrupt handler knows the precise memory configuration when it is called.

Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described in section E.4.

If the interrupt was not caused by a BRK, the handler checks for interrupts that it knows how to handle (for example, a properly initialized mouse) and handles them.

Depending on the state of the system, it either ignores other interrupts, or passes them to a user's interrupt handling routine whose address is stored at $3FE and $3FF of main memory.

After handling an interrupt itself, or after the user's handler returns (with an RTI), the built-in interrupt handler restores the memory configuration, and then does an RTI to restore processing to where it was when the interrupt occurred. Figure E-1 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

Figure E-1. Interrupt-Handling Sequence

Interrupted Program	Processor	Built-in Handler	User's Handler
Program →	Push Address		
	Push Status		
	JMP ($FFFE) →	Save old and set new memory configuration.	
		If BRK, then go to break handler ($FA47). →	
		Our interrupt?	
		NO: Push Address	
		Push Status	
		JMP ($3FE) →	Handle interrupt.
			...
		YES: Handle it.	...
		Restore memory ←	RTI
		configuration.	
	Pull Status ←	RTI	
Program ←	Pull Address		

E.2.1 Saving the Memory Configuration

The built-in interrupt handler saves the state of the system, and sets it to a known state according to these rules:

- If 80STORE and PAGE2 are on, then it switches in Text Page 1 (PAGE2 off) so that main screen holes are accessible.
- It switches in main memory for reading (RAMRD off).
- It switches in main memory for writing (RAMWRT off).
- It switches in ROM addresses $D000-$FFFF for reading (RDLCRAM off).
- It switches in main stack and zero page (ALTZP off).
- It preserves the auxiliary stack pointer, and restores the main stack pointer (see section E.2.2).

Note: Because main memory is switched in, all memory addresses used later in this appendix are in main memory unless otherwise specified.

E.2.2 Managing Main and Auxiliary Stacks

Because the Apple IIc has two stack pages, the firmware has established a convention that allows the system to be run with two separate stack pointers. Two bytes in the auxiliary stack page are to be used as storage for inactive stack pointers: $100 for the main stack pointer when the auxiliary stack is active, and $101 for the auxiliary stack pointer when the main stack is active.

When a program that uses interrupts switches in the auxiliary stack for the first time, it should place the value of the main stack pointer at auxiliary stack address $100, and initialize the auxiliary stack pointer to $FF (the top of the stack). When it subsequently switches from one stack to the other, it should save the current stack pointer before loading the pointer for the other stack.

When an interrupt occurs while the auxiliary stack is switched in, the current stack pointer is stored at $101, and the main stack pointer is retrieved from $100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.

E.3 User's Interrupt Handler at $3FE

Screen hole locations can be set up to indicate that the user's interrupt handler should be called when certain interrupts occur. To use such a routine, place its address at $3FE and $3FF in main memory, low byte first.

The user's interrupt handler should do the following:

- Verify that the interrupt came from the expected source. The following sections describe how this should be done for each interrupt source.
- Handle the interrupt as desired.
- Clear the interrupt, if necessary. The following sections describe how to clear the interrupts.
- Return using an RTI instruction.

If your interrupt handler needs to know the memory configuration at the time of the interrupt, it can check the encoded byte stored four bytes down on the stack. This byte is explained in section E.4.

In general there is no guaranteed response time for interrupts. This is because the system may be doing a disk operation, which could last for several seconds.

Once the built-in interrupt handler has been called, it takes about 250 to 300 microseconds for it to call your interrupt handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.

If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst possible state (80STORE and PAGE2 on, auxiliary memory switched in for reading and writing, auxiliary bank-switched memory page 2 switched in for reading and writing).

E.4 Handling Break Instructions

After the interrupt handler has set the memory configuration, it checks to see if the interrupt was caused by a BRK (opcode $00) instruction. (If it was, bit 4 of the processor status byte is a one). If so, it jumps to a break handling routine, which saves the state of the computer at the time of the break as follows.

Information	Location
Program counter (low byte)	$3A
Program counter (high byte)	$3B
Encoded memory state	$44
Accumulator	$45
X register	$46
Y register	$47
Status register	$48

Finally the break routine jumps to the routine whose address is stored at $3F0 and $3F1.

The encoded memory state in location $44 can be interpreted as follows:

Bit 7	=	0	
Bit 6	=	1	if 80STORE and PAGE2 both on
Bit 5	=	1	if auxiliary RAM switched in for reading
Bit 4	=	1	if auxiliary RAM switched in for writing
Bit 3	=	1	if bank-switched RAM being read
Bit 2	=	1	if bank-switched $D000 page 1 switched in
Bit 1	=	1	if bank-switched $D000 page 2 switched in
Bit 0	=	0	

E.5 Sources of Interrupts

The Apple IIc can receive interrupts from many different sources. Each source is enabled and used slightly differently from the others. There are two basic classes of interrupt sources: those associated with use of the mouse, and those associated with the two 6551 ACIA circuits (the chips that control serial communication).

The interrupts that are associated with the mouse are these:

- An interrupt can be generated when the mouse is moved in the horizontal (X) direction.
- An interrupt can be generated when the mouse is moved in the vertical (Y) direction.
- An interrupt can be generated every 1/60 second. This is called the **vertical blanking** (VBL) interrupt, and is synchronized with a signal used for the video display.
- Using the firmware, interrupts can be generated when the mouse button is pressed.

The interrupts that are associated with the ACIAs are these:

- An interrupt can be generated when a key is pressed. The firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user.
- An interrupt can be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.
- An interrupt can be generated when either ACIA has received a byte of data from its port. The firmware can use this interrupt to buffer data or it can pass the interrupt on to the user.
- An interrupt can be generated when pin 5 of either serial port changes state (device ready/not ready to accept data). When the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data.
- An interrupt can be generated when either ACIA is ready to accept another character to be transmitted. When the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data.
- An interrupt is generated when the keyboard strobe is cleared. The firmware absorbs this interrupt.

E.6 Firmware Handling of Interrupts

The following sections discuss the various sources of interrupts and how they should be used in conjunction with the built-in interrupt handler.

E.6.1 Firmware for Mouse and VBL

As described in Chapter 9, the mouse can be initialized (by the SETMOUSE call) to nine different modes that enable one or more sources of interrupts. In transparent mode, the interrupts are entirely handled by the built-in interrupt handler; the other modes require a user-installed interrupt handler.

When the mouse is initialized, the interrupt vector is copied to addresses $FFFE and $FFFF in main and auxiliary bank-switched RAM. This permits mouse interrupts with any memory configuration.

When the mouse is active, possible sources of interrupts are:

- Mouse movement in the X direction
- Mouse movement in the Y direction
- Change of state of the button
- Rising edge of the vertical blanking signal.

When an interrupt occurs, the built-in interrupt handler determines whether that particular interrupt source was enabled (by the SETMOUSE call). If so, the user's interrupt handler, whose address is stored at $3FE, is called.

The user's interrupt handler should first call SERVEMOUSE to determine the source of the interrupt. This call updates the mouse status byte at $77C and returns with the carry bit clear if mouse movement, button, or vertical blanking was the source of the interrupt.

The values of this mouse status byte at $77C are as follows:

Bit	**1 means that**
3	Interrupt was from vertical blanking
2	Interrupt was from button
1	Interrupt was from mouse movement

If the interrupt was due to mouse movement or button, the user's interrupt handler should then do a call to READMOUSE. This causes the mouse coordinates and status to be updated as follows:

$47C	Low byte of X coordinate
$4FC	Low byte of Y coordinate
$57C	High byte of X coordinate
$5FC	High byte of Y coordinate

$77C Button and movement status

Bit	Means
7	0 = button up; 1 = button down
6	0 = button up on last READMOUSE
	1 = button down on last READMOUSE
5	0 = no movement since last READMOUSE
	1 = movement since last READMOUSE
3-1	always set to 0 (interrupt cleared)

After the interrupt has been handled, the routine should terminate with an RTI.

As already mentioned, interrupts may be missed during disk accesses.

If you turn on mouse interrupts without initializing the mouse, the built-in interrupt handler will absorb the interrupts. If you want to handle mouse interrupts yourself, you must write your own interrupt handler and place vectors to it at addresses $FFFE and $FFFF in bank-switched RAM. Interrupts will be ignored whenever the $D000-$FFFF ROM is switched in.

E.6.2 Firmware for Keyboard Interrupts

The Apple IIc hardware is able to generate an interrupt when a key is pressed. The firmware is able to buffer up to 128 keystrokes, completely transparently, when properly enabled to do so. It saves them in the second half of page 8 of auxiliary memory. After the buffer is full, subsequent keystrokes are ignored. Because interrupts are only generated when keypresses occur, characters generated by the auto-repeat feature are not buffered. They can, however, be read when the buffer is empty.

Once keyboard buffering has been turned on, the next key should be read by calling RDKEY ($FD0C).

▲ **Warning**
Do not call the buffer reading routine directly. Its entry address will not be the same in future versions of the computer.

The special characters CONTROL-S (stop list) and CONTROL-C (stop Applesoft execution) do not work while keyboard buffering is turned on. A new keystroke, -CONTROL-X, clears the buffer.

Using Keyboard Buffering Firmware

See Chapter 8.

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself.

1. Disable processor interrupts (SEI).
2. Set location $5FA to $80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations $5FF and $6FF to $80. These are pointers to where in the buffer the next keystroke will be stored and where the next will be read from, respectively.
4. Turn on the ACIA for port 2 by setting the low nibble of $C0AA to the value $F. For example:

   ```
   LDA $C0AA     ;read port 2 ACIA command register
   ORA #$0F      ;set low nibble to $F
   STA $C0AA     ;set port 2 ACIA command register
   ```

   If you are using the serial ports at the same time, just set the low bit of $C0AA to 1. This prevents receiver interrupts from being turned off.

   A PR#2 or IN#2 or the equivalent will shut off keyboard interrupts.
5. Enable processor interrupts (CLI).

Using Keyboard Interrupts Through Firmware

Keyboard interrupts are received through the ACIA for port 2. They can be enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location $5FA to $C0. This tells the firmware to identify a keystroke interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 2 by setting the low nibble of $C0AA to the value $F. For example:

```
LDA $C0AA     ;read port 2 ACIA command register
ORA #$0F      ;set low nibble to $F
STA $C0AA     ;set port 2 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify the keyboard as the interrupt source by reading location $4FA. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 2, bit 7 is set. If the interrupt was caused by a keystroke, bit 6 is set and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting $4FA to 0.

E.6.3 Using External Interrupts Through Firmware

Pin 9 of the external disk drive connector (EXTINT) can be used to generate interrupts through the ACIA for port 1. It can be used as a source of interrupts (on a high-to-low transition) if enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location $5F9 to $C0. This tells the firmware to identify an external interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 1 by setting the low nibble of $C09A to the value $F. For example:

```
LDA $C09A     ;read port 1 ACIA command register
ORA #$0F      ;set low nibble to $F
STA $C09A     ;set port 1 ACIA command register
```

4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify this interrupt by reading location $4F9. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 1, bit 7 is set. If the interrupt was caused by the external interrupt line, bit 6 is clear and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting $4F9 to 0.

E.6.4 Firmware for Serial Interrupts

The Apple IIc hardware is able to generate interrupts both when the ACIA receives data and when it is ready to send data. The built-in interrupt handler responds to incoming data only. The firmware is able to buffer up to 128 incoming bytes of serial data from either serial port. After the buffer is full, data is ignored. Only one port can be buffered at a time.

The following sections assume that the serial port to be buffered is already initialized, as explained in Chapter 8.

Using Serial Buffering Transparently

Serial buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself, as follows:

1. Disable processor interrupts (SEI).
2. Set location $4FF to $C1 to buffer port 1, or to $C2 to buffer port 2.
3. Set locations $57F and $67F to 0. These are pointers to the next byte in the buffer to be used and the next character to be read from the buffer, respectively.
4. Turn on the ACIA for the port by setting the low nibble of $C09A for port 1 or $C0AA for port 2 to $D. For example:

   ```
   LDA $C09A     ;read port 1 ACIA command register
   AND $F0       ;clear low nibble
   ORA #$0D      ;set low nibble to $D
   STA $C09A     ;set port 1 ACIA command register
   ```

   The 0 in bit 1 of the command register enables receiver interrupts, thus an interrupt is generated when a byte of data is received.
5. Enable processor interrupts (CLI).

See Chapter 8.

When serial port buffering is thus enabled, normal reads from the serial port firmware fetch data from the buffer rather than directly from the ACIA.

Using Serial Interrupts Through Firmware

It is also possible to use the firmware to call the user interrupt handler whenever a byte of data is read by the ACIA. In this mode buffering is not performed by the firmware.

1. Disable processor interrupts (SEI).
2. Set location $4FF to a value other than $C1 or $C2.
3. Turn on the ACIA for the port by setting the low nibble of $C09A for port 1 or $C0AA for port 2 to $D. For example:

   ```
   LDA $C09A      ;read port 1 ACIA command register
   AND $F0        ;clear low nibble
   ORA #$0D       ;set low nibble to $D
   STA $C09A      ;set port 1 ACIA command register
   ```

   The 0 in bit 1 of the command register enables receiver interrupts, thus an interrupt is generated when a byte of data is received.
4. Enable processor interrupts (CLI).

When a serial port is thus enabled, the user's interrupt handler is called each time the port receives a byte of data. The status byte saved by the firmware ($4F9 for port 1; $4FA for port 2) has the high bit set if the interrupt occurred on that port. Bit 3 is set if the interrupt was due to a received byte of data.

The interrupt handler should clear the interrupt by clearing bits 7 and 3 of that port's status byte ($4F9 for port 1; $4FA for port 2).

Transmitting Serial Data

The serial firmware does not implement buffering for serial output. Instead it waits for two conditions to be true before transmitting a character:

- The ACIA's transmit register must be ready to accept a character. This is true if bit 4 of the ACIA's status register is 1.
- The device must signal that it is ready to accept data. This is true if bit 5 of the ACIA's status register is 0. Bit 5 is 0 if pin 5 of the port's connector is also 0.

When the serial firmware is active, a change of state on pin 5 of that port generates an interrupt. That interrupt is absorbed, but the data remains in bit 5 of the status register. Interrupts from the ACIA's transmit register are normally disabled.

A Loophole in the Firmware

So that programs can make use of interrupts on the ACIAs without affecting mouse interrupt handling, there is a tiny loophole purposely left in the built-in interrupt handler. If transmit interrupts are enabled on the ACIA—that is, if bits 3, 2, and 0 of the ACIA's command register have the values 0, 1, and 1, respectively—then control is passed to the user's interrupt handler if the interrupt is not intended for the mouse (movement, button, or VBL).

This means that you can write more sophisticated serial interrupt handling routines than the limited firmware space could provide (such as printer spooling). The firmware will still set memory to its standard state, handle mouse interrupts, and restore memory after your routine is finished.

When you receive the interrupt, neither ACIA's status register has been read. You are fully responsible for checking for interrupts on both ACIAs, determining which of the four interrupt sources on each ACIA caused the interrupt, and how to handle them. Refer to the 6551 specification for more details. The built-in firmware itself is an excellent example of how interrupts on the ACIA can be handled.

E.7 Bypassing the Interrupt Firmware

The following sections give further details on using interrupts on the Apple IIc computer without using the built-in interrupt handler.

E.7.1 Using Mouse Interrupts Without the Firmware

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If $D000-$FFFF ROM is ever switched in, the built-in interrupt handler will absorb the mouse interrupts. Tables E-1 and E-2 show how to activate and read mouse interrupts without using the firmware. Remember to disable interrupts (SEI) before enabling mouse interrupts, then turn them on when done (CLI).

Table E-1. *Activating Mouse Interrupts*

To Activate Interrupts On	Enable IOU Access	Select Source	Enable Source	Disable IOU Access
Mouse X (rising edge)	STA $C079	STA $C05C	STA $C059	STA $C078
Mouse X (falling edge)	STA $C079	STA $C05D	STA $C059	STA $C078
Mouse Y (rising edge)	STA $C079	STA $C05E	STA $C059	STA $C078
Mouse Y (falling edge)	STA $C079	STA $C05F	STA $C059	STA $C078
VBL	STA $C079		STA $C05B	STA $C078

Table E-2. *Reading Mouse Interrupts*

To Read Interrupts From	Read Direction (A.S.A.P.)	Determine Source	Handle It	Return
Mouse X	LDA $C066	LDA $C015 (bit 7=1 if true)	...	RTI
Mouse Y	LDA $C067	LDA $C017 (bit 7=1 if true)	...	RTI
VBL		LDA $C019 (bit 7=1 if true)	...	RTI

The mouse direction data read from $C066 and $C067 is guaranteed to be valid for at least 40 microseconds. Average duration is at least 200 microseconds. This means you should read the direction as soon as possible.

E.7.2 Using ACIA Interrupts Without the Firmware

To use ACIA interrupts without the firmware, you must set your own interrupt vectors. If the $D000-$FFFF ROM is ever switched in, the built-in interrupt handler will handle the interrupt as determined by certain mode bytes (section E.6.1).

When writing your serial interrupt handler, refer to Figures 11.31 through 11-33 and to the Synertek 6551 ACIA specification. As shown in Chapter 11, the ACIA's have the following connections:

Port 1: DSR line connected to the EXTINT line on the external disk port

DCD line connected to pin 5 of Port 1 connector

Port 2: DSR line goes high when a key is pressed

DCD line connected to pin 5 of Port 2 connector

The ACIA registers have the following addresses:

Port 1:	Data Register	=	$C098
	Status Register	=	$C099
	Command Register	=	$C09A
	Control Register	=	$C09B
Port 2:	Data Register	=	$C0A8
	Status Register	=	$C0A9
	Command Register	=	$C0AA
	Control Register	=	$C0AB

Apple II Series Differences

This appendix compares the Apple IIc to the Apple IIe, Apple II Plus, and Apple II.

F.1 Overview

This appendix does not contain an exhaustive list of differences. However, it does include those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more models from the Apple II Series.

As an overview, the differences between the Apple II series computers can be expressed as a series of equations: this computer equals that one plus or minus certain features.

Note: The following *equations* are merely an overview of what each model of Apple II Series is with respect to its predecessor. These equations are in terms of functional equivalence, not strict equality. For example,

Apple IIe = Apple II Plus + Apple Language Card

does not mean there is an actual language card or slot—just that the one machine functions as if it were the other with such a card (with its connector) in a slot.

Apple II Plus	=	II	+	Autostart ROM
			+	Applesoft firmware
			+	48K RAM standard
			-	Old Monitor ROM
			-	Integer BASIC firmware
Apple IIe	=	II Plus	+	Apple Language Card (with 16K of RAM)
			+	80-column (enhanced) video firmware
			+	built-in diagnostics
			+	full ASCII keyboard
			+	internal power light
			+	FCC approval
			+	improved back panel
			+	9-pin back panel game connector
			+	auxiliary slot (with possibility of 80-column text card
			+	extra 64K RAM)
			-	slot 0
Apple IIc	=	IIe	+	extended 80-column text card
			+	80/40 switch
			+	keyboard switch
			+	disk light
			+	disk controller port
			+	disk drive
			+	mouse port
			+	serial printer port
			+	serial communication port
			+	built-in port firmware
			+	video expansion connector
			-	removable cover
			-	slots 1 to 7
			-	auxiliary slot
			-	internal power light
			-	cassette I/O connectors
			-	internal game I/O connector (hence no game *output)*
			-	auxiliary video pin
			-	monitor cassette support

F.1.1 Type of CPU

The CPU in the Apple II and II Plus is the 6502. The Apple IIe uses a 6502A. The Apple IIc uses the 65C02: this is a redesigned CMOS CPU that has 27 new instructions, new addressing modes, and for some instructions a differing execution scheme and machine cycle counts (see Appendix A). Programs written for the Apple IIc will run on the earlier machines only if they do not contain instructions unique to the 65C02, or depend on instruction cycle times that differ.

F.1.2 Machine Identification

Identification of Apple II series computers is as follows:

Machine	$FBB3	$FBC0	$FB1E
Apple II	$38		
Apple II Plus	$EA		
Apple IIe	$06	$EA	
Apple IIc	$06	$00	
Apple III in Apple II Emulation Mode	$EA		$8A

Any future Apple II series computer or ROM release will have different values in these locations. Machine identification routines are available from Apple Vendor Technical Support.

With regard to ProDOS, its MACHID byte, at location $BF98 on the global page, will have bit 3 set to 0 if the computer is an Apple II, II Plus, IIe, or III, and a 1 if the computer is not one of these machines. In addition, for an Apple IIc, bits 7 and 6 are set to binary 10.

Bits 7 and 6 set to binary 10 indicate that a computer is Apple IIe and IIc compatible, regardless of the value of bit 3.

F.2 Memory Structure

This section compares the memory organization of the Apple IIc with that of the Apple II, II Plus, and IIe. These machines differ in RAM space, ROM space, slot or port address space, and hardware page use.

F.2.1 Amount and Address Ranges of RAM

The Apple II could have as little as 4K of RAM at the time of purchase, and could be upgraded to as much as 48K of RAM, following a procedure described in the *Apple II Reference Manual*.

The Apple II Plus has 48K of RAM ($0000 through $BFFF) as a standard feature. With the addition of an Apple Language Card, a 48K Apple II or II Plus could be expanded to have 64K of RAM.

The Apple IIe has a full 64K of RAM. The top 12K addresses overlap with the ROM addresses $D000 through $FFFF. There is an additional `bank-switched` area of 4K from $D000 through $DFFF. This arrangement is equivalent to an Apple II Plus with an Apple Language Card installed. A program selects between the RAM and ROM address spaces and between the $Dxxx banks by changing soft switches located in memory.

With an Extended 80-Column Text Card installed in its auxiliary slot, an Apple IIe has an additional 64K of RAM available, although no more than half of the 128K of RAM space is available at any given time. Soft switches located in memory control these address space selections.

The RAM in the Apple IIc is equivalent to the RAM in an Apple IIe with an Extended 80-column Card.

F.2.2 Amount and Address Ranges of ROM

The Apple II has 8K of ROM ($E000 through $FFFF), and the Apple II Plus has 12K of ROM ($D000 through $FFFF). Users can plug their own ROMs into the sockets provided. The on-board (as opposed to slot) ROM address range is from $D000 through $FFFF.

The Apple IIe has 16K of ROM, of which it uses 15.75 K (addresses $C100 through $FFFF; page $C0 addresses are for I/O hardware). ROM addresses $C300 through $C3FF (normally assigned to the ROM in a card in slot 3) and $C800 through $CFFF contain 80-column video firmware; ROM addresses $C100 through $C2FF and $C400 through $C7FF (normally assigned to the ROM on cards in slots 1, 2, 4, 5, 6 and 7) contain built-in self-test routines.

A soft switch in RAM controls whether the video firmware or slot 3 card ROM is active. Invoking the self-tests with 🍎-CONTROL-RESET causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc ROM also uses the 15.75 K from $C100 through $FFFF, and its enhanced video firmware has the same entry point addresses as on the Apple IIe. However, there are only rudimentary built-in self-tests, and these do not pre-empt any port firmware space.

In the Apple IIc, addresses $C100 through $CFFF contain I/O and interrupt firmware, addresses $D000 through $F7FF contain the Applesoft BASIC Interpreter, and addresses $F800 through $FFFF contain the Monitor.

F.2.3 Peripheral-Card Memory Spaces

Each Apple IIc port has up to sixteen peripheral-card I/O space locations in main memory on the hardware page (beginning at location $C0s0 + $80 for slot or port s), allocated in the standard Apple II series way (that is, beginning at location $C0s0 + $80 for each slot s).

The peripheral-card ROM space (page $Cs for slot s in the Apple II, II Plus, and IIe) contains the starting and entry-point addresses for port s, but port routines are not limited to their allocated $Cs pages.

The 2K-byte expansion ROM space from $C800 to $CFFF in the Apple IIc is used by the enhanced video firmware and miscellaneous I/O and memory-transfer routines.

The 128 bytes of peripheral-card RAM space or *scratch-pad RAM* (64 screen holes in main memory and their equivalent addresses in auxiliary memory) are reserved for use by the built-in firmware. It is extremely important for the correct operation of Apple IIc firmware that these locations not be altered by software except for the specific purposes described in Chapters 7, 8, and 9, and in Appendix E.

F.2.4 Hardware Addresses

The hardware page (the addresses from $C000 through $C0FF) controls memory selection and input/output hardware characteristics. All input and output (except video output) takes place at one or more hardware page addresses. For the sake of simplicity, this section presents only a general comparison between the Apple IIc on the one hand, and the Apple II, II Plus, and IIe on the other, with respect to most hardware page uses. However, for many characteristics, the Apple IIe and IIc work one way, while the Apple II and II Plus work another.

$C000 to $C00F

On all Apple II series computers, reading any one of these addresses reads the keyboard data and strobe. On the Apple IIe and IIc, writing to each of these addresses turns memory and display switches on and off. Writing to addresses $C006, $C007, $C00A, and $C00B performs ROM selection on the Apple IIe. Writing to these four addresses is reserved on the Apple IIc.

For reading the keyboard, use $C000; reserve $C001 through $C00F.

$C010 to $C01F

On all Apple II series computers, writing to any one of these addresses clears the keyboard strobe. On the Apple IIe and IIc, reading each of these addresses checks the status of a memory or display switch, or the any-key-down flag.

For clearing the keyboard strobe, use $C010; reserve $C011 through $C01F.

Reading $C015 checks the SLOTCXROM switch on the Apple IIe, but it resets the X-movement interrupt (XINT) on the Apple IIc. Similarly, reading $C017 checks the SLOTC3ROM switch on the Apple IIe, but it resets the Y-movement interrupt (YINT) on the Apple IIc.

Reading $C019 checks the current state of vertical blanking (VBL) on the Apple IIe, but it resets the latched vertical blanking interrupt (VBLINT) on the Apple IIc.

$C020 to $C02F

On the Apple II, II Plus, and IIe, reading any address $C02x toggles the cassette output signal. On the Apple IIc, both reading from and writing to these locations are reserved.

$C030 to $C03F

On all Apple II series computers, reading an address of the form $C03x toggles the speaker. For full Apple II series compatibility, toggle the speaker using $C030, and reserve $C031 through $C03F.

On the Apple IIc, writing to these addresses is explicitly reserved.

$C040 to $C04F

On the Apple II, II Plus, and IIe, reading any address of the form $C04x triggers the Utility Strobe. The Apple IIc has no Utility Strobe.

On the Apple IIc, addresses $C044 through $C047 are explicitly reserved, and reading or writing any address from $C048 through $C04F resets both the X and Y interrupts (XINT and YINT).

$C050 to $C05F

Addresses $C050 through $C057 work the same on the Apple IIc as on the Apple IIe: they turn the TEXT, MIXED, PAGE2 and HIRES switches on and off.

On the Apple IIe, addresses $C058 through $C05F turn the annunciator outputs on and off. On an Apple IIe with a revision B main logic board, an Apple Extended 80-Column Text Card, and a jumper installed on the card, reading locations $C05E and $C05F set and clear double-high-resolution display mode.

On the Apple IIc, if the IOUDIS switch is on, both reading from and writing to addresses $C058 through $C05D are reserved, and addresses $C05E and $C05F set and clear double-high-resolution display (as on the Apple IIe equipped as described in the preceding paragraph). If the IOUDIS switch is off, then addresses $C058 through $C05F control various characteristics of mouse and vertical blanking interrupts (Table 9-2).

$C060 to $C06F

On the Apple IIc, writing to any address of the form $C06x is reserved, and reading addresses $C068 through $C06F is reserved.

Reading addresses $C061 and $C062 is the same as on the Apple IIe (switch inputs and Apple keys). Reading addresses $C064 and $C065 is the same as on all other Apple II series computers (analog inputs 0 and 1).

On the Apple IIc, address $C063 bit 7 is 1 if the mouse switch is not pressed, and 0 if it is pressed, so that software looking for the *shift-key mod* (used on Apple II, II Plus, and IIe with some text cards) will *find* it and display lowercase correctly. If by chance the mouse button is pressed when the software checks location $C063, it will appear that the shift-key mod is not present.

On the Apple IIc, address $C060 is used for reading the state of the 80/40 switch; on the Apple II, II Plus, and IIe, this address is for reading cassette input.

The Apple IIc has two, rather than four, analog (*paddle*) inputs. Addresses $C066 and $C067 are used for reading the mouse X and Y direction bits.

$C070 to $C07F

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form $C07x triggered the (analog input) paddle timers.

On the Apple IIc, only address $C070 is to be used for that one function. Addresses $C071 through $C07D are explicitly reserved. The results of reading from or writing to addresses $C07E and $C07F are described in Table 5-8.

$C080 to $C08F

On the Apple IIe and IIc, accessing addresses in this range selects different combinations of bank-switched memory banks. However, addresses $C084 through $C087 duplicate the functions of the four addresses preceding them, and addresses $C08C through $C08F do also. These eight addresses are explicitly reserved on the Apple IIc.

$C090 to $C0FF

On the Apple II, II Plus, and IIe, each group of 16 addresses of the form $C080 + $s0 is allocated to an interface card (if present) in slot s.

On the Apple IIc, addresses corresponding to slots 1, 2, 3, 4 and 6 are allocated to a serial interface card, communication interface card, 80-column text card, mouse interface card, and disk controller card, respectively. All other addresses in this range are reserved.

F.2.5 Monitors

The older models of the Apple II and Apple II Plus included a different version of the System Monitor from the one built into more recent models (and the Apple IIe and IIc). The older version, called the Monitor ROM, had the same standard I/O subroutines as the newer Autostart ROM, but a few of their features were different; for example, there were no arrow keys for vertical cursor motion.

When you start the Apple IIc with a DOS or BASICS disk and it loads Integer BASIC into the bank-switched area in RAM, it loads the old Monitor along with it. When you type `INT` from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type `FP` to switch back to Applesoft, which uses the new Monitor in ROM, or activate the 80-column firmware.

F.3 I/O in General

Apple IIc I/O is different from I/O on the Apple II, II Plus, and IIe in three important respects: the possibility of direct memory access (DMA) transfers, the presence or absence of slots, and the presence or absence of built-in interrupt handling.

F.3.1 DMA Transfers

The Apple II, II Plus, and IIe allow DMA transfers, because both the address and the data bus are available at the slots. No true DMA transfer is possible with the Apple IIc because neither bus is available at any of the back-panel connectors.

F.3.2 Slots Versus Ports

The Apple II and II Plus have eight identical slots; the Apple IIe has seven identical slots plus a 60-pin auxiliary slot for video, add-on memory and test cards. The Apple IIc has no slots; instead, it has built-in hardware and firmware that are functional equivalents of slots with cards in them (and back-panel connectors). These are called **ports** on the Apple IIc.

F.3.3 Interrupts

Interrupts on the Apple IIc are described in Appendix E.

The Apple IIc is the first computer in the Apple II Series to have built-in interrupt-handling capabilities.

F.4 Keyboard

Both keyboard layout and character sets vary in the Apple II series computers. The major keyboard difference in the Apple II Series is that the Apple IIe and IIc have full ASCII keyboards, while the Apple II and II Plus do not.

F.4.1 Keys

The Apple II and II Plus have identical 52-key keyboards. The Apple IIe and Apple IIc keyboards have the same 63-key full ASCII keyboard layout, with new and repositioned keys and characters as compared to the Apple II and II Plus. While the Apple II and II Plus have a REPT key, the IIe and IIc have an auto-repeat feature built into each character key.

Some Apple II and Apple II Plus machines have a slide switch inside the case, under the keyboard edge of the cover, for selecting whether or not RESET works without CONTROL. On the Apple IIe and Apple IIc, there is no choice: CONTROL-RESET works, and RESET alone does not.

The Apple IIc and IIe have an ○ and a key; the Apple II and II Plus do not have these two keys.

The captions on several keys—ESC, TAB, CONTROL, SHIFT, CAPS-LOCK, DELETE, RETURN, and RESET—can vary: on the Apple II and II Plus some are abbreviated or missing; on the Apple IIc all keycaps are lowercase italic; on international models, some captions are replaced by symbols (Appendix G).

The Apple IIc has two switches that the other models do not have. One switch is for changing between 40-column and 80-column display, the other is for selecting keyboard layout (Sholes versus Dvorak on USA models), or both keyboard layout and character set (on international models).

The position of the power-on light differs on the Apple II and II Plus, Apple IIe, and Apple IIc. The Apple IIc has a disk-use light as well.

F.4.2 Character Sets

The Apple II and II Plus keyboard character sets are the same. They are described in the *Apple II Reference Manual.*

The Apple IIe and Apple IIc keyboard character sets are the same: full ASCII. The standard (Sholes) layout and key assignments are described in the *Apple IIe Reference Manual.* The Dvorak layout and key assignments are described in Chapter 4 and Appendix G of this manual.

To change between the two available keyboard layouts requires modification to the main logic board on the Apple IIe, but only toggling of the keyboard switch on the Apple IIc.

Apple Computer, Inc. manufactures fully localized models (power supply and character sets) of both the Apple IIe and the Apple IIc. However, there are minor variations in keyboard layout, even among early and late productions models of the same machine. For further details, refer to Appendix G of this manual or to the Apple IIe *Supplement to the Owner's Manual.*

F.5 Speaker

The Apple IIc has two speaker features that the three previous models do not have. They are a two-channel, but monaural, mini-phone jack for headphones—which disconnects the internal speaker when something is plugged into it—and a volume control.

F.6 Video Display

This section discusses the general differences between Apple IIc video display capabilities and those of the other computers in the series. Note however that as new ROMs become available for the Apple IIe, many differences between these two machines will vanish.

F.6.1 Character Sets

The Apple II and II Plus display only uppercase characters, but they display them in three ways: normal, inverse, and flashing. The Apple IIc and IIe can display uppercase characters in all three ways, and they can display lowercase characters in the normal way. This combination is called the **primary character set**.

The Apple IIc and IIe have another character set, called the **alternate character set**, that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 5. You can switch character sets at any time by means of the ALTCHAR soft switch, also described in Chapter 5.

Flashing display must not be used with the enhanced video firmware active. Use it in 40-column mode with the enhanced video firmware turned off; otherwise, strange displays may result, such as MouseText characters appearing in place of uppercase letters.

To be compatible with some software, you have to switch the Apple IIc keyboard to uppercase by pressing (CAPS LOCK).

F.6.2 MouseText

MouseText characters (Chapter 5) are available on every Apple IIc, and on any Apple IIe that has had its ROMs appropriately upgraded, if necessary.

F.6.3 Vertical Blanking

A signal called **vertical blanking** indicates when a display device should stop projecting dots until the display mechanism returns from the bottom of the screen to the top to make another pass. During this interval, a program can make changes to display memory pages, and thus provide a smooth, flicker-free transition to a new display.

On the Apple IIe, vertical blanking (VBL) is a signal whose level must be polled. (VBL is not available to software on the Apple II or II Plus.) On the Apple IIc, vertical blanking is an interrupt (VBLINT) that occurs on the trailing edge of the active-low VBL signal. Programs intended to run on all Apple II series computers must take this difference into account.

F.6.4 Display Modes

All models have 40-column text mode, low-resolution graphics mode, high-resolution graphics mode, and mixed graphics and text modes. The Apple IIe (revision B motherboard) with an Apple Extended 80-Column Text Card, and the Apple IIc have double-high-resolution graphics mode also.

F.7 Disk I/O

The Apple II, II Plus, and IIe can support up to six (four is the recommended maximum) disk drives attached in controller cards plugged into slots 6, 5, and 4. The Apple IIc supports up to two disk drives: its built-in drive (treated as slot 6 drive 1), and one external disk drive (treated as slot 6 drive 2; also treated as slot 7 drive 1 under ProDOS) for external-drive startup purposes.

F.8 Serial I/O

The Apple IIc serial ports (ports 1 and 2) are similar to Super Serial Cards installed in slots 1 and 2 of an Apple IIe. The serial port commands are a slightly modified subset of Super Serial Card commands. This subset includes all the commands supported by the earlier Apple Serial Interface Card and Communication Card.

F.8.1 Serial Ports Versus Serial Cards

There are several important differences between Apple IIc serial ports and other Apple II series computers with serial cards installed in them.

Apple IIc serial ports have no switches. Instead, initial values are moved from firmware locations into auxiliary memory when the power is turned on. Changes made to these values in auxiliary memory remain in effect until the power is turned off. Pressing 🍎-CONTROL-RESET does not change them.

When the port itself is turned on (with an IN or PR command), the initial values in auxiliary memory are placed in the main memory screen holes assigned to the port. These characteristics can be changed by the port commands. The changed characteristics remain in effect until the port is turned off and then on again (with PR and IN commands).

The command syntax for the Apple IIc ports also differs from the syntax for serial cards. A separate command character, CONTROL-A or CONTROL-I, must precede each individual port command, whereas several commands to a serial card can be strung together between the command character and a carriage return character.

The letters used for some of the commands have been changed from those used with the Super Serial Card (such as *S* instead of *B* for sending a BREAK signal). Each serial port command letter is unique to simplify command interpretation.

Changing the command character from CONTROL-A to CONTROL-I, or vice versa, makes the Super Serial Card change from communication mode to printer mode and back; this is not the case with Apple IIc serial ports. With the Apple IIc, use the *System Utilities Disk* to change modes.

Super Serial Card commands support several functions that Apple IIc serial port commands don't support: masking incoming line feed after carriage return; translating incoming characters, such as changing lowercase to uppercase (for the benefit of the Apple II or II Plus); delaying after sending carriage return, line feed, or form feed; ignoring keyboard input, and so on.

Following a CONTROL-I nnnN command, the Apple IIc automatically generates carriage return after nnn characters; with the Super Serial Card, you need to turn this on with CONTROL-I C.

F.8.2 Serial I/O Buffers

The communication port firmware uses auxiliary memory page 8 as an input and output buffer. By doing so, the firmware can keep up with higher baud rates. It can also *hide* data from the Monitor, Applesoft, and other system software.

Programs written for the Apple IIe or IIc can, of course, store information in auxiliary memory page 8. However, such information will be destroyed when the communication port is activated.

F.9 Mouse and Hand Controls

The DB-9 back-panel connector on the Apple IIc is used for both the mouse and hand controls. On the Apple IIc, the DB-9 connector supports hand controls only. On the Apple IIe, the mouse must use the connector on the interface card.

F.9.1 Mouse Input

The Apple IIc provides built-in firmware support for a mouse connected to the DB-9 mouse and hand control connector. Apple IIc mouse support includes mouse movement and button interrupts (and vertical blanking interrupts for synchronization with the display); Apple IIe mouse support relies on polling VBL instead of vertical blanking interrupts.

As a result of how interrupts are handled on the two machines, the mouse firmware routine calls function somewhat differently for the Apple IIc and Apple IIe. However, using the calls in the manner described in Chapter 9 ensures mouse support compatibility between the two machines.

The ratio of mouse movement to cursor movement is different on the Apple IIc than it is on the Apple IIe.

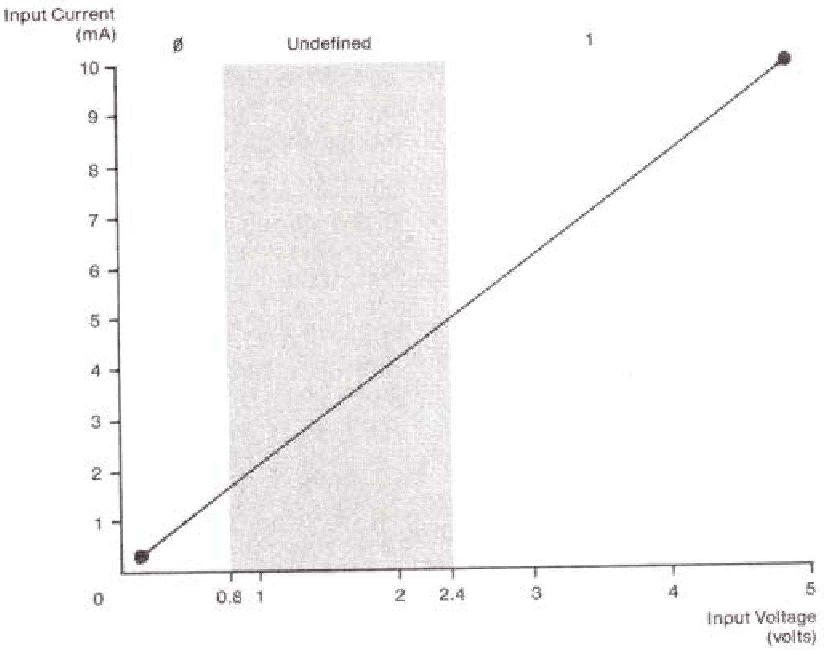
F.9.2 Hand Control Input and Output

The Apple II, II Plus, and IIe have a 16-pin game I/O connector inside the case that supports three switch inputs, four analog (paddle) inputs, and four annunciator outputs. The Apple IIe and Apple IIc have a DB-9 back-panel connector that supports the three switch inputs and two paddle inputs (plus two more on the internal GAME I/O connector of the Apple II, II Plus, and IIe).

The Apple IIc does not support the four annunciator outputs.

The voltage-current curve for hand controls differs for the Apple IIc compared with that of the Apple II, II Plus, and IIe. Compare Figure F-1 with Figure 11-42. This was done so the hardware would support identifiable mouse and hand control signals using the same circuits.

Figure F-1. Apple II, II Plus, and IIe Hand Control Signals



The paddle timing circuit on the Apple II Plus is slightly different than the one on the Apple IIe and IIc. On the Apple IIe and IIc the 100 ohm fixed resistor is betwen the NE556 discharge lead and the capacitor; the variable resistor in the paddle is connected directly to the capacitor. On the Apple II Plus, the capacitor is conected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.

F.10 Cassette I/O

The Apple II, II Plus, and IIe all have cassette input and output jacks, memory locations, and monitor support. The Apple IIc does not.

F.11 Hardware

Besides the different microprocessors used in various models in the Apple II series (section F.1.1), there are important differences in power specifications and custom chips.

F.11.1 Power

The power supplies for the Apple II, II Plus, and IIe are essentially the same. The floor transformer and voltage converter for the Apple IIc have smaller capacity for current and heat dissipation. Therefore, it is important to observe the load limits specified in each of the reference manuals.

F.11.2 Custom Chips

The Apple IIe custom chips (Memory Management Unit and Input/Output Unit) replaced dozens of Apple II Plus chips, and added the functionality of dozens more. The Apple IIc has custom MMU and IOU chips, too, but they represent different *bonding options*, and so their pin assignments are not compatible.

In addition, the Apple IIc has a custom General Logic Unit (GLU), Timing Generator (TMG), and Disk Controller Unit (IWM). The Apple IIc has two hybrid units (AUD and VID) for audio and video amplification.

USA and International Models

appendix g

This appendix repeats some of the keyboard information given in Chapter 4 for the two USA keyboard layouts for easy comparison with the other layouts available. Following these there is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

G.1 Keyboard Layouts and Codes

Each of the following subsections has a keyboard illustration and a table of the codes that result from the possible keystrokes. Note, however, that Table G-1 is the basic table of keystrokes and their codes. For simplicity, subsequent tables (up to Table G-6) list only the keystrokes and codes that differ from those in Table G-1.

For example, pressing the A key produces *a* (hexadecimal 61); pressing SHIFT-A produces uppercase *A* (hexadecimal 41); pressing CONTROL-A or CONTROL-SHIFT-A produces *SOH* (the ASCII Start Of Header control character, hexadecimal 01). You can tell that this key has the same effect on all keyboards, from the fact that nothing appears in Tables G-2 through G-7 for that key.

A quick way to find out which characters in the ASCII set change on international keyboards is to check Table G-7. In fact, only a few of them change. The pairing of characters on keys varies more.

Note: On all but the French and Italian keyboards, CAPS-LOCK affects only keys that can produce both lowercase letters (with or without an accent) and their uppercase equivalents. With these keys, CAPS-LOCK down is equivalent to holding down SHIFT, resulting in uppercase instead of lowercase. If a key produces only a lowercase version of an accented letter, then CAPS-LOCK does not affect it.

On the French and Italian keyboards, CAPS-LOCK shifts all the keys. Furthermore, on the French keyboard, when CAPS-LOCK is down, the SHIFT key undoes the shifting.

Note: The shapes and arrangement of keys in Figures G-1 and G-2 follow the ANSI (American National Standards Institute) standard, which is used mainly in North and South America. The shapes and arrangement of keys in Figure G-3 follows the ISO (International Standards Organization) standard used in Europe and elsewhere.

The only differences between the ANSI and ISO versions of the USA keyboard are

- The shapes of three keys: the left SHIFT key, CAPS-LOCK, and RETURN.
- The resulting repositioning of two keys (| and ¯) in Figures G-1 and G-3.
- For some countries, there are arrow symbols on TAB, CAPS-LOCK, RETURN, and the two SHIFT keys (as shown in Figure G-3).

G.1.1 USA Standard (Sholes) Keyboard

Figure G-1 shows the Standard (Sholes) keyboard as it is laid out for USA models of the Apple IIc with the keyboard switch up. Table G-1 lists the ASCII codes resulting from all simple and combination keystrokes on this keyboard.

Figure G-1. USA Standard or Sholes Keyboard (Keyboard Switch Up)

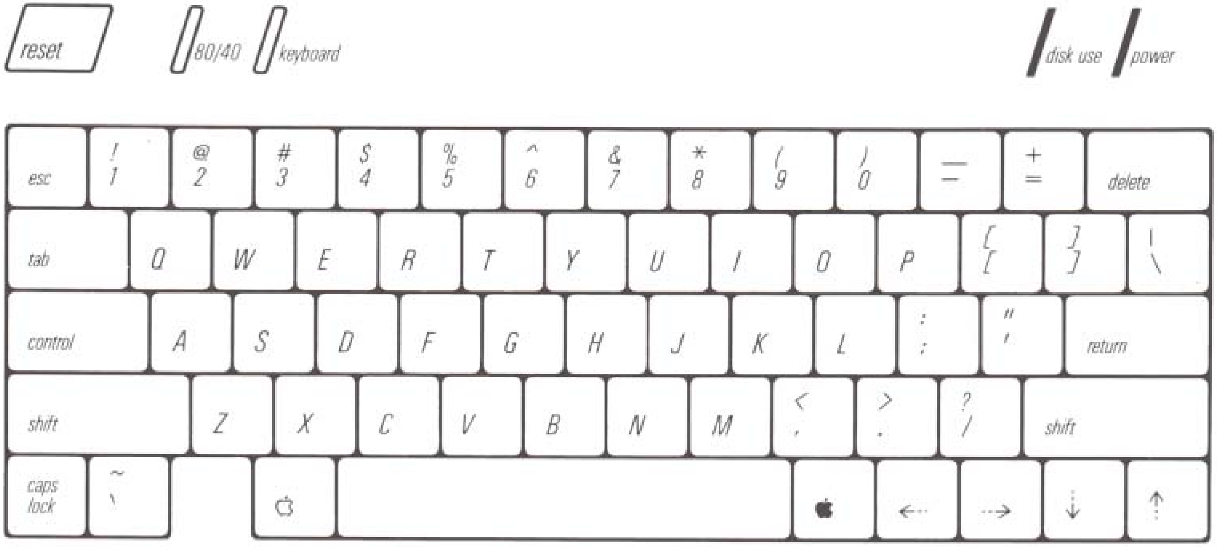


Table G-1. *Keys and ASCII Codes. Codes are shown here in hexadecimal; to find the decimal equivalents, use Table G-7.*

	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	**Hex**	**Char**	**Hex**	**Char**	**Hex**	**Char**	**Hex**	**Char**
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
←	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
↓	0A	LF	0A	LF	0A	LF	0A	LF
↑	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
→	15	NAK	15	NAK	15	NAK	15	NAK
ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
'"	27	'	27	'	22	"	22	"
,<	2C	,	2C	,	3C	<	3C	<
-_	2D	-	1F	US	5F	_	1F	US
.>	2E	.	2E	.	3E	>	3E	>
/?	2F	/	2F	/	3F	?	3F	?
0)	30	0	30	0	29	)	29	)
1!	31	1	31	1	21	!	21	!
2@	32	2	00	NUL	40	@	00	NUL
3#	33	3	33	3	23	#	23	#
4$	34	4	34	4	24	$	24	$
5%	35	5	35	5	25	%	25	%
6^	36	6	1E	RS	5E	^	1E	RS
7&	37	7	37	7	26	&	26	&
8*	38	8	38	8	2A	*	2A	*
9(	39	9	39	9	28	(	28	(
;:	3B	;	3B	;	3A	:	3A	:
=+	3D	=	3D	=	2B	+	2B	+
[{	5B	[	1B	ESC	7B	{	1B	ESC
\\|	5C	\	1C	FS	7C	|	1C	FS
]}	5D	]	1D	GS	7D	}	1D	GS
`~	60	`	60	`	7E	~	7E	~

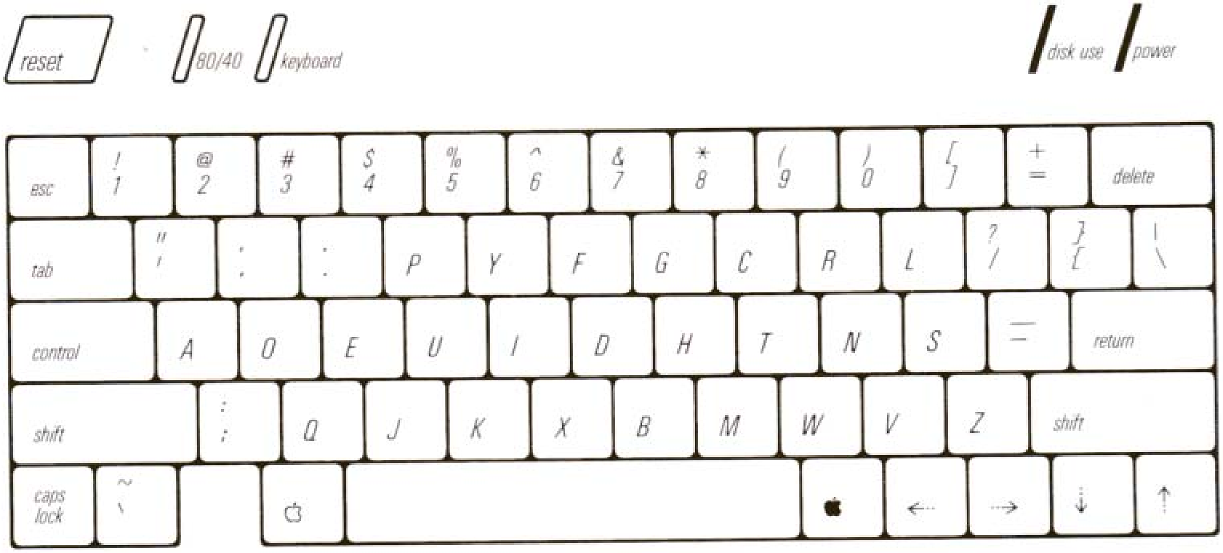
Table G-1—Continued. *Keys and ASCII Codes. Codes are shown here in hexadecimal; to find the decimal equivalents, use Table G-7.*

	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	**Hex**	**Char**	**Hex**	**Char**	**Hex**	**Char**	**Hex**	**Char**
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

G.1.2 USA Simplified (Dvorak) Keyboard

Figure G-2 shows the Dvorak layout of the USA keyboard. Characters are paired up on keys in exactly the same way as on the USA Standard keyboard; only individual key positions are changed. In fact, you can change the keycap arrangement to match Figure G-2, lock the keyboard switch in its down position, and have a working Dvorak keyboard. All keystrokes produce the same ASCII codes as those shown in Table G-1.

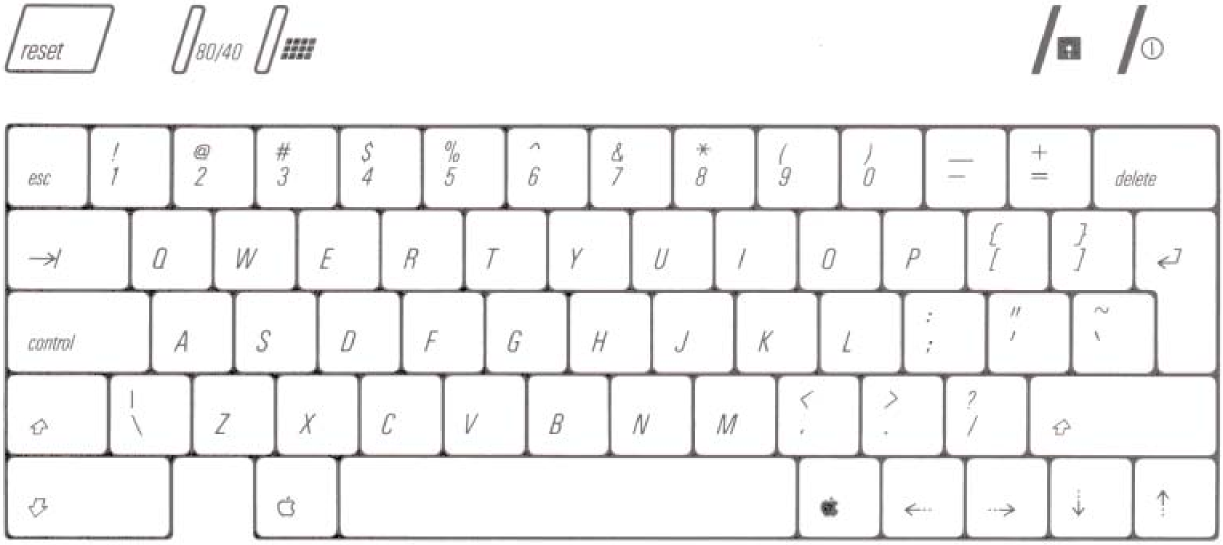
Figure G.2. *USA Simplified or* Dvorak *Keyboard (Keyboard Switch Down)*



G.1.3 ISO Layout of USA Keyboard

Figure G-3 shows the layout of the keyboard of all ISO European keyboards (except the Italian keyboard) when the keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table G-1.

Figure G-3. *ISO Version of USA Standard Keyboard (Keyboard Switch Up)*



G.1.4 English Keyboard

With the keyboard switch up, the English model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the English model keyboard layout is as shown in Figure G-4. The change in ASCII code production (from that in Table G-1) is shown in Table G-2.

The only changed character is the substitution of the British pound-sterling symbol (£) for the cross-hatch symbol (#) on the shifted 3-key.

Figure G-4. English Keyboard (Keyboard Switch Down)

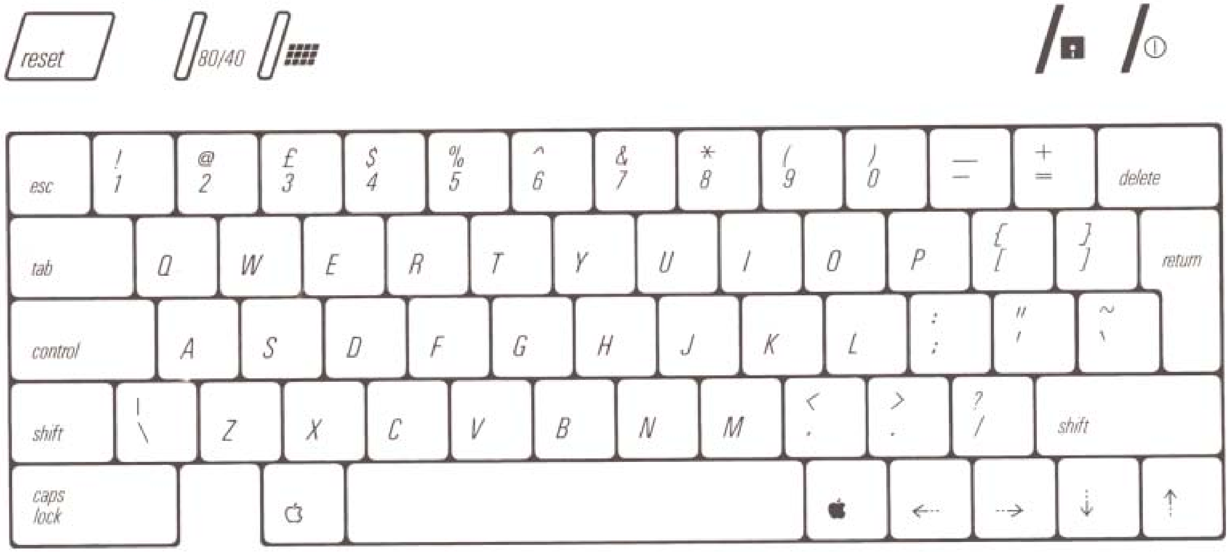


Table G-2. English Keyboard Code Differences From Table G-1

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
	Hex	Char	Hex	Char	Hex	Char	Hex	Char
3£	33	3	33	3	23	£	23	#

G.1.5 French and Canadian Keyboards

With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure G-3, and the Canadian is as shown in Figure G-1. On both models, keystrokes produce the ASCII codes shown in Table G-1.

Note: On the French keyboard, (CAPS-LOCK) shifts to the upper characters on all keys. With (CAPS-LOCK) on, (SHIFT) "unshifts" to the lower character on any key pressed with it.

With the keyboard switch down, the French model keyboard layout is as shown in Figure G-5, and the Canadian model keyboard layout is as shown in Figure G-6. The changes in ASCII code production (from that in Table G-1) are shown in Table G-3.

Figure G-5. *French Keyboard (Keyboard Switch Down)*

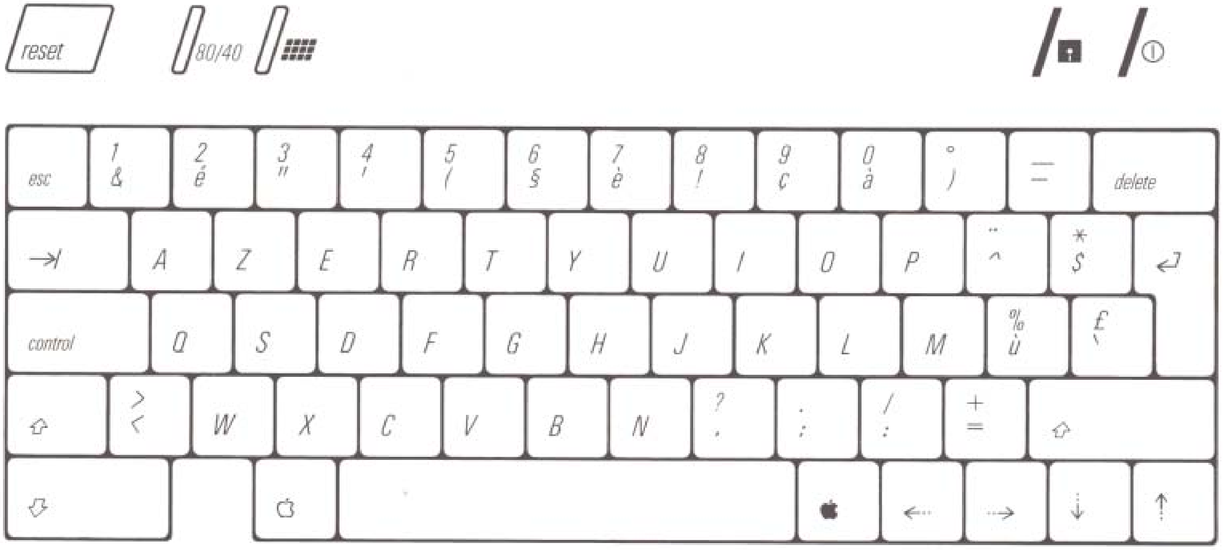


Figure G-6. *Canadian Keyboard (Keyboard Switch Down)*

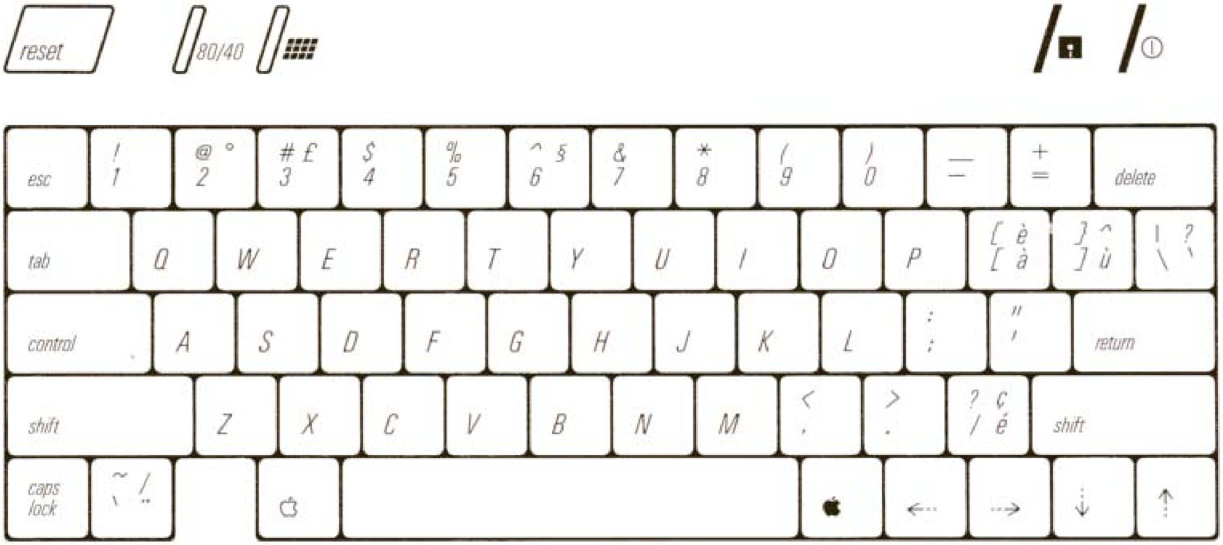


Table G-3. *French and Canadian Keyboard Code Differences From Table G-1*

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
&1	26	&	26	&	31	1	31	1
é2	7B	é	7B	é	32	2	32	2
"3	22	"	22	"	33	3	33	3
'4	27	'	27	'	34	4	34	4
(5	28	(	28	(	35	5	35	5
§6	5D	§	1D	GS	36	6	1D	GS
è7	7D	è	7D	è	37	7	37	7
!8	21	!	21	!	38	8	38	8
ç9	5C	ç	1C	FS	39	9	1C	FS
à0	40	à	00	NUL	30	0	00	NUL
)°	29	)	1B	ESC	5B	°	1B	ESC
^¨	5E	^	1E	RS	7E	¨	1E	RS
$*	24	$	24	$	2A	*	2A	*
ù%	7C	ù	7C	ù	25	%	25	%
`£	60	`	60	`	23	£	23	£
<>	3C	<	3C	<	3E	>	3E	>
,?	2C	,	2C	,	3F	?	3F	?
;.	3B	;	3B	;	2E	.	2E	.
:/	3A	:	3A	:	2F	/	2F	/

G.1.6 German Keyboard

With the keyboard switch up, the German model of the Apple IIc keyboard layout :s as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the German model keyboard layout is as shown in Figure G-7. The change in ASCII code production (from that in Table G-1) is shown in Table G-4.

Figure G-7. *German Keyboard (Keyboard Switch Down)*

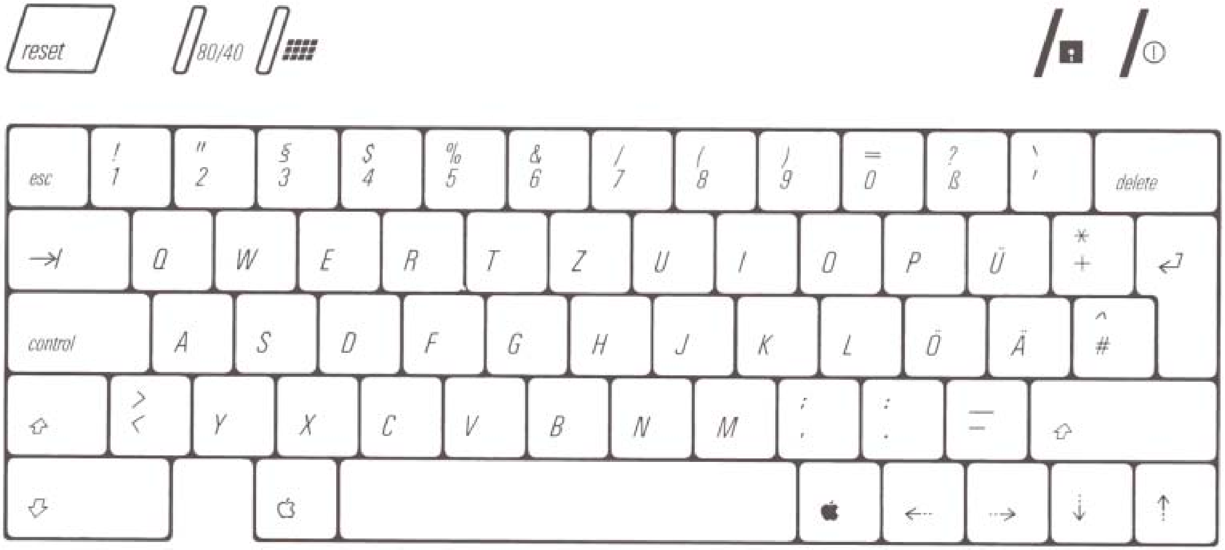


Table G-4. *German Keyboard Code Differences From Table G-1*

Key	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
	Hex	Char	Hex	Char	Hex	Char	Hex	Char
2"	32	2	32	2	22	"	22	"
3§	33	3	00	NUL	40	§	00	NUL
6&	36	6	36	6	26	&	26	&
7/	37	7	37	7	2F	/	2F	/
8(	38	8	38	8	28	(	28	(
9)	39	9	39	9	29	)	29	)
0=	30	0	30	0	3D	=	3D	=
ß ?	7E	ß	7E	ß	3F	?	3F	?
Ü	7D	Ü	1D	GS	5D	Ü	1D	GS
+*	2B	+	2B	+	2A	*	2A	*
Ö	7C	Ö	1C	FS	5C	Ö	1C	FS
Ä	7B	Ä	1B	ESC	5B	Ä	1B	ESC
#^	23	#	1E	RS	5E	^	1E	RS
<>	3C	<	3C	<	3E	>	3E	>
,;	2C	,	2C	,	3B	;	3B	;
.:	2E	.	2E	.	3A	:	3A	:

G.1.7 Italian Keyboard

With the keyboard switch down, the Italian model keyboard layout is as shown in Figure G-8. The change in ASCII code production (from that in Table G-1) is shown in Table G-5.

With the keyboard switch up, the Italian model keyboard produces exactly the same ASCII codes for each key, but what is displayed differs for the ten characters shown in Table G-5 or Table G-7.

Figure G-8. *Italian Keyboard (Keyboard Switch Down)*

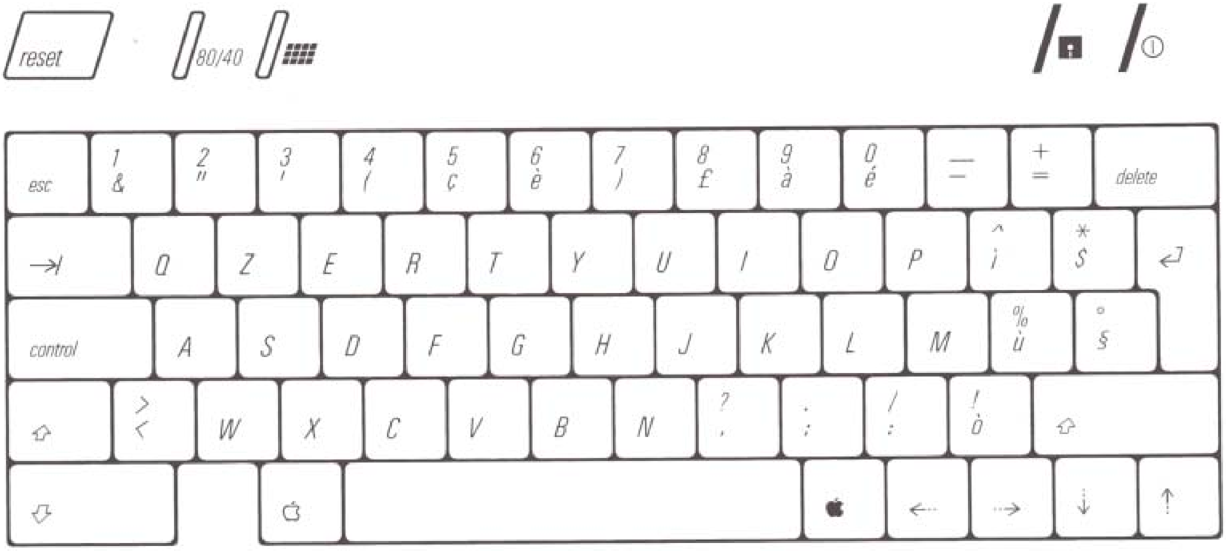


Table G-5. *Italian Keyboard Code Differences From Table G-1*

	Key Alone		CONTROL + Key		SHIFT + Key		Both + Key	
Key	Hex	Char	Hex	Char	Hex	Char	Hex	Char
&1	26	&	26	&	31	1	31	1
"2	22	"	22	"	32	2	32	2
'3	27	'	27	'	33	3	33	3
(4	28	(	28	(	34	4	34	4
ç5	5C	ç	1C	FS	35	5	1C	FS
è6	7D	è	7D	è	36	6	36	6
)7	29	)	29	)	37	7	37	7
£8	23	£	23	£	38	8	38	8
à9	7B	à	7B	à	39	9	39	9
é0	5D	é	1D	GS	30	0	1D	GS
ì ^	7E	ì	1E	RS	5E	^	1E	RS
$*	24	$	24	$	2A	*	2A	*
ù%	60	ù	60	ù	25	%	25	%
§°	40	§	00	NUL	5B	°	1B	ESC
<>	3C	<	3C	<	3E	>	3E	>
,?	2C	,	2C	,	3F	?	3F	?
;.	3B	;	3B	;	2E	.	2E	.
:/	3A	:	3A	:	2F	/	2F	/
ò!	7C	ò	7C	ò	21	!	21	!

G.1.8 Western Spanish Keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the IIc keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Western Spanish model keyboard layout is as shown in Figure G-9. The change in ASCII code production (from that in Table G-1) is shown in Table G-6.

Figure G-9. Western Spanish Keyboard (Keyboard Switch Down)

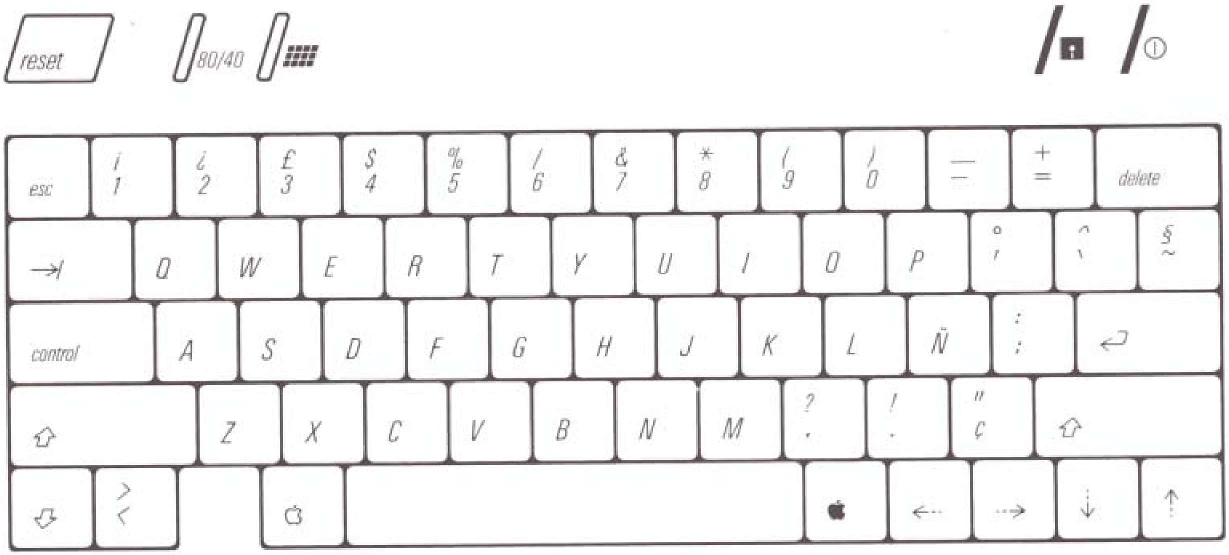


Table G-6. Western Spanish Keyboard Code Differences From Table G-1

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
2"	32	2	32	2	22	"	22	"
3£	33	3	33	3	23	£	23	£
6&	36	6	00	NUL	26	&	00	NUL
7/	37	7	37	7	2F	/	2F	/
8(	38	8	38	8	28	(	28	(
9)	39	9	39	9	29	)	29	)
0=	30	0	30	0	3D	=	3D	=
'?	27	'	27	'	3F	?	3F	?
`¿	60	`	60	`	5D	¿	5D	¿
~^	7E	~	1E	RS	5E	^	1E	RS
+*	2B	+	1B	ESC	2A	*	1B	ESC
Ñ	7C	Ñ	1C	FS	5C	Ñ	1C	FS
ç¡	7D	ç	7D	ç	5B	¡	5B	¡
°§	7B	°	00	NUL	40	§	00	NUL
<>	3C	<	1E	RS	3E	>	1E	RS
,;	2C	,	2C	,	3B	;	3B	;
.:	2E	.	2E	.	3A	:	3A	:

G.2 ASCII Character Sets

Table G-7 lists the ASCII (American National Standard Code for Information Interchange) codes that the Apple IIc uses, as well as the decimal and hexadecimal equivalents. Where there are differences between character sets, a circled number in the main table refers to a column in the lower part of the table.

Table G-7. ASCII Code Equivalents

ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX
NUL	00	00	SP	32	20	•②	64	40	•⑦	96	60
SOH	01	01	!	33	21	A	65	41	a	97	61
STX	02	02	"	34	22	B	66	42	b	98	62
ETX	03	03	•⓪	35	23	C	67	43	c	99	63
EOT	04	04	•①	36	24	D	68	44	d	100	64
ENQ	05	05	%	37	25	E	69	45	e	101	65
ACK	06	06	&	38	26	F	70	46	f	102	66
BEL	07	07	'	39	27	G	71	47	g	103	67
BS	08	08	(	40	28	H	72	48	h	104	68
HT	09	09	)	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	;	59	3B	•③	91	5B	•⑧	123	7B
FS	28	1C	<	60	3C	•④	92	5C	•⑨	124	7C
GS	29	1D	=	61	3D	•⑤	93	5D	•⑩	125	7D
RS	30	1E	>	62	3E	•⑥	94	5E	•⑪	126	7E
US	31	1F	?	63	3F	—	95	5F	DEL	127	7F

•	⓪	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
Hexadecimal	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
English (USA)	#	$	@	[	\	]	^	`	{	|	}	~
English (UK)	£	$	@	[	\	]	^	`	{	|	}	~
Deutsch	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Français	£	$	à	°	ç	§	^	`	é	ù	è	¨
Italiano	£	$	§	°	ç	é	^	ù	à	ò	è	ì
Españolo	£	$	§	¡	Ñ	¿	^	`	°	ñ	ç	~

G.3 Certifications

In the countries where they are applicable, these certifications replace the USA FCC Class B notice printed on the inside front cover of this manual. The safety instructions apply to all countries.

G.3.1 Radio Interference

This product is designed to comply with specification VDE 0871/6.78, Radio Frequency Interference Suppression of Radio Frequency Equipment, Level B.

G.3.2 Product Safety

This product is designed to meet the requirements of safety standard IEC 380, Safety of Electrically Energized Office Machines.

G.3.3 Important Safety Instructions

This equipment is intended to be electrically grounded. This product is equipped with a plug having a third (grounding) pin. This plug will fit only into a grounding-type alternating current outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

G.4 Power Supply Specifications

The basic specifications of the power supply furnished with the Apple IIc for use in Europe and other countries having 50 Hz alternating current are shown in Table G-8.

Table G-8. *50 Hz Power Supply Specifications*

Line voltage	199 to 255 VAC, 50 Hz
Maximum input power consumption	25 W
Supply voltage	+15 V DC (nominal)
Supply current	1.2 A (nominal)

Conversion Tables

This appendix briefly discusses bits and bytes and what they can represent. It also contains conversion tables for hexadecimal to decimal and negative decimal, for low-resolution display dot patterns, display color values, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

H.1 Bits and Bytes

This section discusses the relationships between bit values and their position within a byte. The following are some rules of thumb regarding the 65C02.

- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIc are listed in Table H-1.

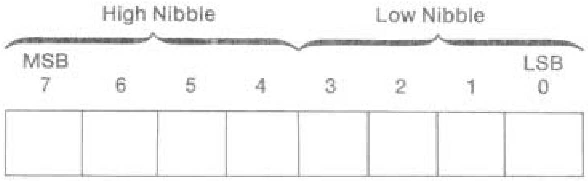
Table H-1. What a Bit Can Represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear†	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

† Sometimes ambiguously termed *reset* .

- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits comprise a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only ten of the sixteen digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure H-1).

Figure H-1. Bits, Nibbles, and Bytes



	7	6	5	4	3	2	1	0
Hexadecimal	$80	$40	$20	$10	$08	$04	$02	$01
Decimal	128	64	32	16	8	4	2	1

Binary	Hexadecimal	Decimal
0000	$0	0
0001	$1	1
0010	$2	2
0011	$3	3
0100	$4	4
0101	$5	5
0110	$6	6
0111	$7	7
1000	$8	8
1001	$9	9
1010	$A	10
1011	$B	11
1100	$C	12
1101	$D	13
1110	$E	14
1111	$F	15

- One byte can represent any of 16 x 16 or 256 values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- One memory position in the Apple IIc contains one eight-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables H-6 through H-13 list some of the ways bytes are commonly interpreted.

- Two bytes make a word. The sixteen bits of a word can represent any one of 256 x 256 or 65536 different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65536 (64K) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

H.2 Hexadecimal and Decimal

Use Table H-2 for conversion of hexadecimal and decimal numbers.

Table H-2. *Hexadecimal/Decimal Conversion*

Digit	$x000	$0x00	$00x0	$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

Examples:

```
$3C = ?

$30 = 48
$0C = 12
--------

$3C = 60
```

```
$FD47 = ?

$F000 = 61440
$ D00 =  3328
$  40 =    64
$   7 =     7
-------------
$FD47 = 64839
```

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have zero left. Add up the hexadecimal numbers.

Example:

```
16215 = $ ?

16215 - 12288 = 3927     12288 = $7000
 3927 -  3840 =   87      3840 = $ F00
   87 -    80 =    7        80 = $  50
            7                7 = $   7
                         -------------
                         16215 = $7F57
```

H.3 Hexadecimal and Negative Decimal

If a number is larger than decimal 32767, Applesoft BASIC allows and Integer BASIC requires that you use the negative-decimal equivalent of the number. Table H-3 is set up to make it easy for you to convert a hexadecimal number directly to a negative decimal number.

Table H-3. *Decimal to Negative Decimal Conversion*

Digit	$x000	$$0x00	$$00x0	$$000x
F	0	0	0	-1
E	-4096	-256	-16	-2
D	-8192	-512	-32	-3
C	-12288	-768	-48	-4
B	-16384	-1024	-64	-5
A	-20480	-1280	-80	-6
9	-24576	-1536	-96	-7
8	-28672	-1792	-112	-8
7		-2048	-128	-9
6		-2304	-144	-10
5		-2560	-160	-11
4		-2816	-176	-12
3		-3072	-192	-13
2		-3328	-208	-14
1		-3584	-224	-15
0		-3840	-240	-16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (zeros included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative decimal number.

Example:

```
$C010 = - ?

$C000:   -12288
$ 000:   - 3840
$   10:  -  224
$    0:  -   16
-------------
$C010    -16368
```

To convert a negative-decimal number directly to a positive decimal number, add it to 65536. (This addition ends up looking like subtraction.)

Example:

```
-151 = + ?

65536 + (-151) = 65536 - 151 = 65385
```

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive decimal number, then use Table H-2.

H.4 Graphics Bits and Pieces

Table H-4 is a quick guide to the hexadecimal values corresponding to 7-bit high-resolution patterns on the display screen. Since the bits are displayed in reverse order, it takes some calculation to determine these values. Table H-4 should make it easy.

The *x* represents bit 7. Zeros represent bits that are off; ones bits that are on. Use the first hexadecimal value if bit 7 is to be off, and the second if it is to be on.

For example, to get bit pattern 00101110, use $3A; for 10101110, use $BA.

Table H-4. *Hexadecimal Values for High-Resolution Dot Patterns*

Bit pattern	(x=0)	(x=1)
x0000000	$00	$80
x0000001	$40	$C0
x0000010	$20	$A0
x0000011	$60	$E0
x0000100	$10	$90
x0000101	$50	$D0
x0000110	$30	$B0
x0000111	$70	$F0
x0001000	$08	$88
x0001001	$48	$C8
x0001010	$28	$A8
x0001011	$68	$E8
x0001100	$18	$98
x0001101	$58	$D8
x0001110	$38	$B8
x0001111	$78	$F8
x0010000	$04	$84
x0010001	$44	$C4
x0010010	$24	$A4
x0010011	$64	$E4
x0010100	$14	$94
x0010101	$54	$D4
x0010110	$34	$B4
x0010111	$74	$F4
x0011000	$0C	$8C
x0011001	$4C	$CC
x0011010	$2C	$AC
x0011011	$6C	$EC
x0011100	$1C	$9C
x0011101	$5C	$DC
x0011110	$3C	$BC
x0011111	$7C	$FC

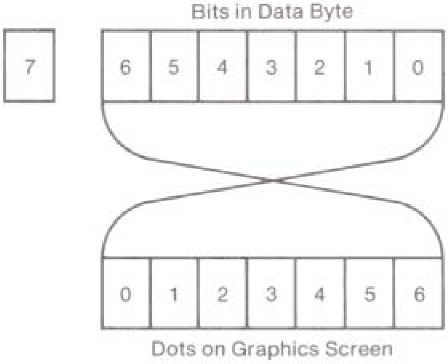


Table H-4—Continued. *Hexadecimal Values for High-Resolution Dot Patterns*

Bit pattern	(x=0)	(x=1)
x0100000	$02	$82
x0100001	$42	$C2
x0100010	$22	$A2
x0100011	$62	$E2
x0100100	$12	$92
x0100101	$52	$D2
x0100110	$32	$B2
x0100111	$72	$F2
x0101000	$0A	$8A
x0101001	$4A	$CA
x0101010	$2A	$AA
x0101011	$6A	$EA
x0101100	$1A	$9A
x0101101	$5A	$DA
x0101110	$3A	$BA
x0101111	$7A	$FA
x0110000	$06	$86
x0110001	$46	$C6
x0110010	$26	$A6
x0110011	$66	$E6
x0110100	$16	$96
x0110101	$56	$D6
x0110110	$36	$B6
x0110111	$76	$F6
x0111000	$0E	$8E
x0111001	$4E	$CE
x0111010	$2E	$AE
x0111011	$6E	$EE
x0111100	$1E	$9E
x0111101	$5E	$DE
x0111110	$3E	$BE
x0111111	$7E	$FE
x1000000	$01	$81
x1000001	$41	$C1
x1000010	$21	$A1
x1000011	$61	$E1
x1000100	$11	$91
x1000101	$51	$D1
x1000110	$31	$B1
x1000111	$71	$F1
x1001000	$09	$89
x1001001	$49	$C9
x1001010	$29	$A9
x1001011	$69	$E9
x1001100	$19	$99
x1001101	$59	$D9
x1001110	$39	$B9
x1001111	$79	$F9

Table H-4—Continued. *Hexadecimal Values for High-Resolution Dot Patterns*

Bit pattern	(x=0)	(x=1)
x1010000	$05	$85
x1010001	$45	$C5
x1010010	$25	$A5
x1010011	$65	$E5
x1010100	$15	$95
x1010101	$55	$D5
x1010110	$35	$B5
x1010111	$75	$F5
x1011000	$0D	$8D
x1011001	$4D	$CD
x1011010	$2D	$AD
x1011011	$6D	$ED
x1011100	$1D	$9D
x1011101	$5D	$DD
x1011110	$3D	$BD
x1011111	$7D	$FD
x1100000	$03	$83
x1100001	$43	$C3
x1100010	$23	$A3
x1100011	$63	$E3
x1100100	$13	$93
x1100101	$53	$D3
x1100110	$33	$B3
x1100111	$73	$F3
x1101000	$0B	$8B
x1101001	$4B	$CB
x1101010	$2B	$AB
x1101011	$6B	$EB
x1101100	$1B	$9B
x1101101	$5B	$DB
x1101110	$3B	$BB
x1101111	$7B	$FB
x1110000	$07	$87
x1110001	$47	$C7
x1110010	$27	$A7
x1110011	$67	$E7
x1110100	$17	$97
x1110101	$57	$D7
x1110110	$37	$B7
x1110111	$77	$F7
x1111000	$0F	$8F
x1111001	$4F	$CF
x1111010	$2F	$AF
x1111011	$6F	$EF
x1111100	$1F	$9F
x1111101	$5F	$DF
x1111110	$3F	$BF
x1111111	$7F	$FF

H.5 Peripheral Identification Numbers

Many Apple products now use Peripheral Identification Numbers (called **PIN numbers**) as shorthand for serial device characteristics. The Apple II Series *System Utilities Disk* presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table H-5 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

Example:

252/1111 means:

Communication mode
8 data bits, 1 stop bit
300 baud (bits per second)

No parity
Do not echo output to display
No linefeed after carriage return
Do not generate carriage returns

Table H-5. PIN Numbers

x x x / x x x x

1 = Printer Mode
2 = Communication Mode *

1 = 6 data bits, 1 stop bit
2 = 6 data bits, 2 stop bits
3 = 7 data bits, 1 stop bit
4 = 7 data bits, 2 stop bits
5 = 8 data bits, 1 stop bit
6 = 8 data bits, 2 stop bits

1 = 110 bits per second
2 = 300 bits per second
3 = 1200 bits per second
4 = 2400 bits per second
5 = 4800 bits per second
6 = 9600 bits per second
7 = 19200 bits per second

1 = No parity
2 = Even parity (total on = even)
3 = Odd parity (total on = odd)
4 = MARK parity (parity bit = 1)
5 = SPACE parity (parity bit = 0)

1 = Do not echo output on screen
2 = Echo output on screen

1 = Do not generate LF after CR
2 = Generate LF after CR

1 = Do not generate CR *
2 = Generate CR after 40 characters
3 = Generate CR after 72 characters
4 = Generate CR after 80 characters
5 = Generate CR after 132 characters

* If you select Communication Mode, then seventh digit must equal 1. This value is supplied automatically when you use the UUD.

H.6 Eight-Bit Code Conversions

Tables H-6 through H-13 show the entire ASCII character set twice: once with the high bit off, and once with it on. Here is how to interpret these tables.

- The **Binary** column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *H*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *H*.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity H, 01001000 for an even-parity H.
- The **ASCII Char** column gives the ASCII character name.
- The **Interpretation** column spells out the meaning of special symbols and abbreviations, where necessary.
- The **What to Type** column indicates what keystrokes generate the ASCII character (where it is not obvious). The numbers between columns refer to footnotes.
- The columns marked **Pri** and **Alt** indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

  Note that the values $40 through $5F (and $C0 through $DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so (section 5.2.2), or if the firmware is bypassed.

> **Note:** The primary and alternate displayed character sets in Tables H-6 through H-13 are the result of firmware mapping. The CHARGEN ROM actually contains only one character set. The firmware mapping procedure is described in section 3.36.

Table H-6. *Control Characters, High Bit Off*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0000000	0	$00	NUL	Blank (null)	CONTROL-@	@	@
0000001	1	$01	SOH	Start of Header	CONTROL-A	A	A
0000010	2	$02	STX	Start of Text	CONTROL-B	B	B
0000011	3	$03	ETX	End of Text	CONTROL-C	C	C
0000100	4	$04	EOT	End of Transm.	CONTROL-D	D	D
0000101	5	$05	ENQ	Enquiry	CONTROL-E	E	E
0000110	6	$06	ACK	Acknowledge	CONTROL-F	F	F
0000111	7	$07	BEL	Bell	CONTROL-G	G	G
0001000	8	$08	BS	Backspace	CONTROL-H or ←	H	H
0001001	9	$09	HT	Horizontal Tab	CONTROL-I or TAB	I	I
0001010	10	$0A	LF	Line Feed	CONTROL-J or ↓	J	J
0001011	11	$0B	VT	Vertical Tab	CONTROL-K or ↑	K	K
0001100	12	$0C	FF	Form Feed	CONTROL-L	L	L
0001101	13	$0D	CR	Carriage Return	CONTROL-M or RETURN	M	M
0001110	14	$0E	SO	Shift Out	CONTROL-N	N	N
0001111	15	$0F	SI	Shift In	CONTROL-O	O	O
0010000	16	$10	DLE	Data Link Escape	CONTROL-P	P	P
0010001	17	$11	DC1	Device Control 1	CONTROL-Q	Q	Q
0010010	18	$12	DC2	Device Control 2	CONTROL-R	R	R
0010011	19	$13	DC3	Device Control 3	CONTROL-S	S	S
0010100	20	$14	DC4	Device Control 4	CONTROL-T	T	T
0010101	21	$15	NAK	Neg. Acknowledge	CONTROL-U or →	U	U
0010110	22	$16	SYN	Synchronization	CONTROL-V	V	V
0010111	23	$17	ETB	End of Text Blk.	CONTROL-W	W	W
0011000	24	$18	CAN	Cancel	CONTROL-X	X	X
0011001	25	$19	EM	End of Medium	CONTROL-Y	Y	Y
0011010	26	$1A	SUB	Substitute	CONTROL-Z	Z	Z
0011011	27	$1B	ESC	Escape	CONTROL-[or ESC	[	[
0011100	28	$1C	FS	File Separator	CONTROL-\	\	\
0011101	29	$1D	GS	Group Separator	CONTROL-]	]	]
0011110	30	$1E	RS	Record Separator	CONTROL-^	^	^
0011111	31	$1F	US	Unit Separator	CONTROL-_	_	_

Table H-7. *Special Characters, High Bit Off*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0100000	32	$20	SP	Space	SPACE bar		
0100001	33	$21	!			!	!
0100010	34	$22	"			"	"
0100011	35	$23	#			#	#
0100100	36	$24	$			$	$
0100101	37	$25	%			%	%
0100110	38	$26	&			&	&
0100111	39	$27	'	Closing Quote		'	'
0101000	40	$28	(			(	(
0101001	41	$29	)			)	)
0101010	42	$2A	*			*	*
0101011	43	$2B	+			+	+
0101100	44	$2C	,	Comma		,	,
0101101	45	$2D	-	Hyphen		-	-
0101110	46	$2E	.	Period		.	.
0101111	47	$2F	/			/	/
0110000	48	$30	0			0	0
0110001	49	$31	1			1	1
0110010	50	$32	2			2	2
0110011	51	$33	3			3	3
0110100	52	$34	4			4	4
0110101	53	$35	5			5	5
0110110	54	$36	6			6	6
0110111	55	$37	7			7	7
0111000	56	$38	8			8	8
0111001	57	$39	9			9	9
0111010	58	$3A	:			:	:
0111011	59	$3B	;			;	;
0111100	60	$3C	<			<	<
0111101	61	$3D	=			=	=
0111110	62	$3E	>			>	>
0111111	63	$3F	?			?	?

Table H-8. Uppercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1000000	64	$40	@			*@*	🍎
1000001	65	$41	A			*A*	🍏
1000010	66	$42	B			*B*	➤
1000011	67	$43	C			*C*	⧗
1000100	68	$44	D			*D*	✓
1000101	69	$45	E			*E*	☑
1000110	70	$46	F			*F*	🏃
1000111	71	$47	G			*G*	🏃
1001000	72	$48	H			*H*	←
1001001	73	$49	I			*I*	…
1001010	74	$4A	J			*J*	↓
1001011	75	$4B	K			*K*	↑
1001100	76	$4C	L			*L*	‾
1001101	77	$4D	M			*M*	↵
1001110	78	$4E	N			*N*	█
1001111	79	$4F	O			*O*	⇤
1010000	80	$50	P			*P*	⇥
1010001	81	$51	Q			*Q*	⤓
1010010	82	$52	R			*R*	⤒
1010011	83	$53	S			*S*	|
1010100	84	$54	T			*T*	└
1010101	85	$55	U			*U*	→
1010110	86	$56	V			*V*	▒
1010111	87	$57	W			*W*	▒
1011000	88	$58	X			*X*	⊏
1011001	89	$59	Y			*Y*	⊐
1011010	90	$5A	Z			*Z*	|
1011011	91	$5B	[	Opening Bracket		*[*	◆
1011100	92	$5C	\	Reverse Slant		**	|
1011101	93	$5D	]	Closing Bracket		*]*	⌗
1011110	94	$5E	^	Caret		*^*	⊡
1011111	95	$5F	_	Underline		—	|

Table H-9. *Lowercase Characters, High Bit Off*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1100000	96	$60	`	Opening Quote			`
1100001	97	$61	a			!	a
1100010	98	$62	b			"	b
1100011	99	$63	c			#	c
1100100	100	$64	d			$	d
1100101	101	$65	e			%	e
1100110	102	$66	f			&	f
1100111	103	$67	g			'	g
1101000	104	$68	h			(	h
1101001	105	$69	i			)	i
1101010	106	$6A	j			*	j
1101011	107	$6B	k			+	k
1101100	108	$6C	l			,	l
1101101	109	$6D	m			-	m
1101110	110	$6E	n			.	n
1101111	111	$6F	o			/	o
1110000	112	$70	p			0	p
1110001	113	$71	q			1	q
1110010	114	$72	r			2	r
1110011	115	$73	s			3	s
1110100	116	$74	t			4	t
1110101	117	$75	u			5	u
1110110	118	$76	v			6	v
1110111	119	$77	w			7	w
1111000	120	$78	x			8	x
1111001	121	$79	y			9	y
1111010	122	$7A	z			:	z
1111011	123	$7B	{	Opening Brace		;	{
1111100	124	$7C	|	Vertical Line		<	|
1111101	125	$7D	}	Closing Brace		=	}
1111110	126	$7E	~	Overline (Tilde)		>	~
1111111	127	$7F	DEL	Delete/Rubout		?	DEL

Table H-10. *Control Characters, High Bit On*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10000000	128	$80	NUL	Blank (null)	CONTROL-@	@	@
10000001	129	$81	SOH	Start of Header	CONTROL-A	A	A
10000010	130	$82	STX	Start of Text	CONTROL-B	B	B
10000011	131	$83	ETX	End of Text	CONTROL-C	C	C
10000100	132	$84	EOT	End of Transm.	CONTROL-D	D	D
10000101	133	$85	ENQ	Enquiry	CONTROL-E	E	E
10000110	134	$86	ACK	Acknowledge	CONTROL-F	F	F
10000111	135	$87	BEL	Bell	CONTROL-G	G	G
10001000	136	$88	BS	Backspace	CONTROL-H or ←	H	H
10001001	137	$89	HT	Horizontal Tab	CONTROL-I or TAB	I	I
10001010	138	$8A	LF	Line Feed	CONTROL-J or ↓	J	J
10001011	139	$8B	VT	Vertical Tab	CONTROL-K or ↑	K	K
10001100	140	$8C	FF	Form Feed	CONTROL-L	L	L
10001101	141	$8D	CR	Carriage Return	CONTROL-M or RETURN	M	M
10001110	142	$8E	SO	Shift Out	CONTROL-N	N	N
10001111	143	$8F	SI	Shift In	CONTROL-O	O	O
10010000	144	$90	DLE	Data Link Escape	CONTROL-P	P	P
10010001	145	$91	DC1	Device Control 1	CONTROL-Q	Q	Q
10010010	146	$92	DC2	Device Control 2	CONTROL-R	R	R
10010011	147	$93	DC3	Device Control 3	CONTROL-S	S	S
10010100	148	$94	DC4	Device Control 4	CONTROL-T	T	T
10010101	149	$95	NAK	Neg. Acknowledge	CONTROL-U or →	U	U
10010110	150	$96	SYN	Synchronization	CONTROL-V	V	V
10010111	151	$97	ETB	End of Text Blk.	CONTROL-W	W	W
10011000	152	$98	CAN	Cancel	CONTROL-X	X	X
10011001	153	$99	EM	End of Medium	CONTROL-Y	Y	Y
10011010	154	$9A	SUB	Substitute	CONTROL-Z	Z	Z
10011011	155	$9B	ESC	Escape	CONTROL-[or ESC	[	[
10011100	156	$9C	FS	File Separator	CONTROL-\	\	\
10011101	157	$9D	GS	Group Separator	CONTROL-]	]	]
10011110	158	$9E	RS	Record Separator	CONTROL-^	^	^
10011111	159	$9F	US	Unit Separator	CONTROL-_	_	_

Table H-11. *Special Characters, High Bit On*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10100000	160	$A0	SP	Space	SPACE bar		
10100001	161	$A1	!			!	!
10100010	162	$A2	"			"	"
10100011	163	$A3	#			#	#
10100100	164	$A4	$			$	$
10100101	165	$A5	%			%	%
10100110	166	$A6	&			&	&
10100111	167	$A7	'	Closed Quote (acute accent)		'	'
10101000	168	$A8	(			(	(
10101001	169	$A9	)			)	)
10101010	170	$AA	*			*	*
10101011	171	$AB	+			+	+
10101100	172	$AC	,	Comma		,	,
10101101	173	$AD	-	Hyphen		-	-
10101110	174	$AE	.	Period		.	.
10101111	175	$AF	/			/	/
10110000	176	$B0	0			0	0
10110001	177	$B1	1			1	1
10110010	178	$B2	2			2	2
10110011	179	$B3	3			3	3
10110100	180	$B4	4			4	4
10110101	181	$B5	5			5	5
10110110	182	$B6	6			6	6
10110111	183	$B7	7			7	7
10111000	184	$B8	8			8	8
10111001	185	$B9	9			9	9
10111010	186	$BA	:			:	:
10111011	187	$BB	;			;	;
10111100	188	$BC	<			<	<
10111101	189	$BD	=			=	=
10111110	190	$BE	>			>	>
10111111	191	$BF	?			?	?

Table H-12. Uppercase Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11000000	192	$C0	@			@	@
11000001	193	$C1	A			A	A
11000010	194	$C2	B			B	B
11000011	195	$C3	C			C	C
11000100	196	$C4	D			D	D
11000101	197	$C5	E			E	E
11000110	198	$C6	F			F	F
11000111	199	$C7	G			G	G
11001000	200	$C8	H			H	H
11001001	201	$C9	I			I	I
11001010	202	$CA	J			J	J
11001011	203	$CB	K			K	K
11001100	204	$CC	L			L	L
11001101	205	$CD	M			M	M
11001110	206	$CE	N			N	N
11001111	207	$CF	O			O	O
11010000	208	$D0	P			P	P
11010001	209	$D1	Q			Q	Q
11010010	210	$D2	R			R	R
11010011	211	$D3	S			S	S
11010100	212	$D4	T			T	T
11010101	213	$D5	U			U	U
11010110	214	$D6	V			V	V
11010111	215	$D7	W			W	W
11011000	216	$D8	X			X	X
11011001	217	$D9	Y			Y	Y
11011010	218	$DA	Z			Z	Z
11011011	219	$DB	[	Opening Bracket		[	[
11011100	220	$DC	\	Reverse Slant		\	\
11011101	221	$DD	]	Closing Bracket		]	]
11011110	222	$DE	^	Caret		^	^
11011111	223	$DF	_	Underline		_	_

Table H-13. Lowercase Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11100000	224	$E0	`	Open Quote		`	`
11100001	225	$E1	a			a	a
11100010	226	$E2	b			b	b
11100011	227	$E3	c			c	c
11100100	228	$E4	d			d	d
11100101	229	$E5	e			e	e
11100110	230	$E6	f			f	f
11100111	231	$E7	g			g	g
11101000	232	$E8	h			h	h
11101001	233	$E9	i			i	i
11101010	234	$EA	j			j	j
11101011	235	$EB	k			k	k
11101100	236	$EC	l			l	l
11101101	237	$ED	m			m	m
11101110	238	$EE	n			n	n
11101111	239	$EF	o			o	o
11110000	240	$F0	p			p	p
11110001	241	$F1	q			q	q
11110010	242	$F2	r			r	r
11110011	243	$F3	s			s	s
11110100	244	$F4	t			t	t
11110101	245	$F5	u			u	u
11110110	246	$F6	v			v	v
11110111	247	$F7	w			w	w
11111000	248	$F8	x			x	x
11111001	249	$F9	y			y	y
11111010	250	$FA	z			z	z
11111011	251	$FB	{	Opening Brace		{	{
11111100	252	$FC	|	Vertical Line		|	|
11111101	253	$FD	}	Closing Brace		}	}
11111110	254	$FE	‾	Overline (Tilde)		‾	‾
11111111	255	$FF	DEL	Delete (Rubout)	DELETE	DEL	DEL

Firmware Listings

appendix i

Appendix I comprises a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc.

```
C100:                *********************************
C100:                *
C100:                * Apple //c
C100:                * Video Firmware and
C100:                * Monitor ROM Source
C100:                *
C100:                * COPYRIGHT 1977-1983 BY
C100:                * APPLE COMPUTER, INC.
C100:                *
C100:                * ALL RIGHTS RESERVED
C100:                *
C100:                * S. WOZNIAK             1977
C100:                * A. BAUM                1977
C100:                * JOHN A             NOV 1978
C100:                * R. AURICCHIO       SEP 1981
C100:                * E. BEERNINK            1983
C100:                *
C100:                *********************************
C100:                *
C100:                * ZERO PAGE EQUATES
C100:                *
C100:          0000  LOC0      EQU   $00          ;vector for autostart from disk
C100:          0001  LOC1      EQU   $01
C100:          0020  WNDLFT    EQU   $20          ;left edge of text window
C100:          0021  WNDWDTH   EQU   $21          ;width of text window
C100:          0022  WNDTOP    EQU   $22          ;top of text window
C100:          0023  WNDBTM    EQU   $23          ;bottom+1 of text window
C100:          0024  CH        EQU   $24          ;cursor horizontal position
C100:          0025  CV        EQU   $25          ;cursor vertical position
C100:          0026  GBASL     EQU   $26          ;lo-res graphics base addr.
C100:          0027  GBASH     EQU   $27
C100:          0028  BASL      EQU   $28          ;text base address
C100:          0029  BASH      EQU   $29
C100:          002A  BAS2L     EQU   $2A          ;temp base for scrolling
C100:          002B  BAS2H     EQU   $2B
C100:          002C  H2        EQU   $2C          ;temp for lo-res graphics
C100:          002C  LMNEM     EQU   $2C          ;temp for mnemonic decoding
C100:          002D  V2        EQU   $2D          ;temp for lo-res graphics
C100:          002D  RMNEM     EQU   $2D          ;temp for mnemonic decoding
C100:          002E  MASK      EQU   $2E          ;color mask for lo-res gr.
C100:          002E  FORMAT    EQU   $2E          ;temp for opcode decode
C100:          002F  LENGTH    EQU   $2F          ;temp for opcode decode
C100:          0030  COLOR     EQU   $30          ;color for lo-res graphics
C100:          0031  MODE      EQU   $31          ;Monitor mode
C100:          0032  INVFLG    EQU   $32          ;normal/inverse(/flash)
C100:          0033  PROMPT    EQU   $33          ;prompt character
C100:          0034  YSAV      EQU   $34          ;position in Monitor command
C100:          0035  YSAV1     EQU   $35          ;temp for Y register
C100:          0036  CSWL      EQU   $36          ;character output hook
C100:          0037  CSWH      EQU   $37
C100:          0038  KSWL      EQU   $38          ;character input hook
C100:          0039  KSWH      EQU   $39
C100:          003A  PCL       EQU   $3A          ;temp for program counter
C100:          003B  PCH       EQU   $3B
C100:          003C  A1L       EQU   $3C          ;Monitor temp
C100:          003D  A1H       EQU   $3D          ;Monitor temp
C100:          003E  A2L       EQU   $3E          ;Monitor temp
C100:          003F  A2H       EQU   $3F          ;Monitor temp
```

```
C100:     0040    A3L       EQU   $40          ;Monitor temp
C100:     0041    A3H       EQU   $41          ;Monitor temp
C100:     0042    A4L       EQU   $42          ;Monitor temp
C100:     0043    A4H       EQU   $43          ;Monitor temp
C100:     0044    A5L       EQU   $44          ;Monitor temp
C100:     0045    A5H       EQU   $45          ;Monitor temp
C100:             *
C100:             * Note: In Apple II, //e, both interrupts and BRK destroyed
C100:             * location $45.  Now only BRK destroys $45 (ACC) and it
C100:             * also destroys $44 (MACSTAT).
C100:             *
C100:     0044    MACSTAT   EQU   $44          ;Machine state after BRK
C100:     0045    ACC       EQU   $45          ;Acc after BRK
C100:             *
C100:     0046    XREG      EQU   $46          ;X reg after break
C100:     0047    YREG      EQU   $47          ;Y reg after break
C100:     0048    STATUS    EQU   $48          ;P reg after break
C100:     0049    SPNT      EQU   $49          ;SP after break
C100:     004E    RNDL      EQU   $4E          ;random counter low
C100:     004F    RNDH      EQU   $4F          ;random counter high
C100:             *
C100:             * Value equates
C100:             *
C100:     0006    GOODF8    EQU   $06          ;value of //e, lolly ID byte
C100:     0095    PICK      EQU   $95          ;CONTROL-U character
C100:     009B    ESC       EQU   $9B          ;what ESC generates
C100:             *
C100:             * Characters read by GETLN are placed in
C100:             * IN, terminated by a carriage return.
C100:             *
C100:     0200    IN        EQU   $0200        ;input buffer for GETLN
C100:             *
C100:             * Page 3 vectors
C100:             *
C100:     03F0    BRKV      EQU   $03F0        ;vectors here after break
C100:     03F2    SOFTEV    EQU   $03F2        ;vector for warm start
C100:     03F4    PWREDUP   EQU   $03F4        ;THIS MUST = EOR #$A5 OF SOFTEV+1
C100:     03F5    AMPERV    EQU   $03F5        ;APPLESOFT & EXIT VECTOR
C100:     03F8    USRADR    EQU   $03F8        ;APPLESOFT USR function vector
C100:     03FB    NMI       EQU   $03FB        ;NMI vector
C100:     03FE    IRQLOC    EQU   $03FE        ;Maskable interrupt vector
C100:     0400    LINE1     EQU   $0400        ;first line of text screen
C100:     07F8    MSLOT     EQU   $07F8        ;owner of $C8 space
C100:             *
C100:             * HARDWARE EQUATES
C100:             *
C100:     C000    IOADR     EQU   $C000        ;for IN#, PR# vector
C100:     C000    KBD       EQU   $C000        ;>127 if keystroke
C100:     C000    CLR80COL  EQU   $C000        ;disable 80 column store
C100:     C001    SET80COL  EQU   $C001        ;enable 80 column store
C100:     C002    RDMAINRAM EQU   $C002        ;read from main 48K RAM
C100:     C003    RDCARDRAM EQU   $C003        ;read from alt. 48K RAM
C100:     C004    WRMAINRAM EQU   $C004        ;write to main 48K RAM
C100:     C005    WRCARDRAM EQU   $C005        ;write to alt. 48K RAM
C100:     C008    SETSTDZP  EQU   $C008        ;use main zero page/stack
C100:     C009    SETALTZP  EQU   $C009        ;use alt. zero page/stack
C100:     C00C    CLR80VID  EQU   $C00C        ;disable 80 column hardware
C100:     C00D    SET80VID  EQU   $C00D        ;enable 80 column hardware
```

```
C100:      C00E   CLRALTCHAR EQU   $C00E        ;normal LC, flashing UC
C100:      C00F   SETALTCHAR EQU   $C00F        ;normal inverse, LC; no flash
C100:      C010   KBDSTRB    EQU   $C010        ;turn off key pressed flag
C100:      C011   RDLCBNK2   EQU   $C011        ;>127 if LC bank 2 is in
C100:      C012   RDLCRAM    EQU   $C012        ;>127 if LC RAM read enabled
C100:      C013   RDRAMRD    EQU   $C013        ;>127 if reading main 48K
C100:      C014   RDRAMWRT   EQU   $C014        ;>127 if writing main 48K
C100:      C016   RDALTZP    EQU   $C016        ;>127 if Alt ZP and LC switched in
C100:      C018   RD80COL    EQU   $C018        ;>127 if 80 column store
C100:      C019   RDVBLBAR   EQU   $C019        ;>127 if not VBL
C100:      C01A   RDTEXT     EQU   $C01A        ;>127 if text (not graphics)
C100:      C01B   RDMIX      EQU   $C01B        ;>127 if mixed mode on
C100:      C01C   RDPAGE2    EQU   $C01C        ;>127 if TXTPAGE2 switched in
C100:      C01D   RDHIRES    EQU   $C01D        ;>127 if HIRES is on
C100:      C01E   ALTCHARSET EQU   $C01E        ;>127 if alternate char set in use
C100:      C01F   RD80VID    EQU   $C01F        ;>127 if 80 column hardware in
C100:      C020   TAPEOUT    EQU   $C020        ;what is this??
C100:      C030   SPKR       EQU   $C030        ;clicks the speaker
C100:      C050   TXTCLR     EQU   $C050        ;switch in graphics (not text)
C100:      C051   TXTSET     EQU   $C051        ;switch in text (not graphics)
C100:      C052   MIXCLR     EQU   $C052        ;clear mixed-mode
C100:      C053   MIXSET     EQU   $C053        ;set mixed-mode (4 lines text)
C100:      C054   TXTPAGE1   EQU   $C054        ;switch in text page 1
C100:      C055   TXTPAGE2   EQU   $C055        ;switch in text page 2
C100:      C056   LORES      EQU   $C056        ;low-resolution graphics
C100:      C057   HIRES      EQU   $C057        ;high-resolution graphics
C100:      C058   CLRAN0     EQU   $C058
C100:      C059   SETAN0     EQU   $C059
C100:      C05A   CLRAN1     EQU   $C05A
C100:      C05B   SETAN1     EQU   $C05B
C100:      C05C   CLRAN2     EQU   $C05C
C100:      C05D   SETAN2     EQU   $C05D
C100:      C05E   CLRAN3     EQU   $C05E
C100:      C05F   SETAN3     EQU   $C05F
C100:      C060   RD40SW     EQU   $C060        ;>127 if 40/80 switch in 40 pos
C100:      C061   BUTN0      EQU   $C061        ;open apple key
C100:      C062   BUTN1      EQU   $C062        ;closed apple key
C100:      C064   PADDL0     EQU   $C064        ;read paddle 0
C100:      C070   PTRIG      EQU   $C070        ;trigger the paddles
C100:      C081   ROMIN      EQU   $C081        ;switch in $D000-$FFFF ROM
C100:      C083   LCBANK2    EQU   $C083        ;switch in LC bank 2
C100:      C08B   LCBANK1    EQU   $C08B        ;switch in LC bank 1
C100:      CFFF   CLRROM     EQU   $CFFF        ;switch out $C8 ROMs
C100:      E000   BASIC      EQU   $E000        ;BASIC entry point
C100:      E003   BASIC2     EQU   $E003        ;BASIC warm entry point
C100:             *
C100:      04FB   VMODE      EQU   $4F8+3       ;OPERATING MODE
C100:             *
C100:             * BASIC VMODE BITS
C100:             *
C100:             * 1....... - BASIC active
C100:             * 0....... - Pascal active
C100:             * .0......
C100:             * .1......
C100:             * ..0..... - Print control characters
C100:             * ..1..... - Don't print ctrl chars
C100:             * ...0.... -
C100:             * ...1.... -
```

```
C100:              176 * ....0... - Print control characters
C100:              177 * ....1... - Don't print ctrl chars.
C100:              178 * .....0.. -
C100:              179 * .....1.. -
C100:              180 * ......0. -
C100:              181 * ......1. -
C100:              182 * .......0 - Print mouse characters
C100:              183 * .......1 - Don't print mouse characters
C100:              184 *
C100:        0040  185 M.40       EQU    $40
C100:        0020  186 M.CTL2     EQU    $20           ;Don't print controls
C100:        0008  187 M.CTL      EQU    $08           ;Don't print controls
C100:        0001  188 M.MOUSE    EQU    $01           ;Don't print mouse chars
C100:              189 *
C100:              190 * Pascal Mode Bits
C100:              191 *
C100:              192 * 1....... - BASIC active
C100:              193 * 0....... - Pascal active
C100:              194 * .0......
C100:              195 * .1......
C100:              196 * ..0..... -
C100:              197 * ..1..... -
C100:              198 * ...0.... - Cursor always on
C100:              199 * ...1.... - Cursor always off
C100:              200 * ....0... - GOTOXY n/a
C100:              201 * ....1... - GOTOXY in progress
C100:              202 * .....0.. - Normal Video
C100:              203 * .....1.. - Inverse Video
C100:              204 * ......0. -
C100:              205 * ......1. -
C100:              206 * .......0 - Print mouse chars
C100:              207 * .......1 - Don't print mouse chars
C100:              208 *
C100:        0080  209 M.PASCAL   EQU    $80           ;Pascal active
C100:        0010  210 M.CURSOR   EQU    $10           ;Don't print cursor
C100:        0008  211 M.GOXY     EQU    $08           ;GOTOXY IN PROGRESS
C100:        0004  212 M.VMODE    EQU    $04
C100:              213 *
C100:        0478  214 ROMSTATE   EQU    $478          ;temp store of ROM state
C100:        04F8  215 TEMP1      EQU    $4F8          ;used by CTLCHAR
C100:        0578  216 TEMPA      EQU    $578          ;used by scroll
C100:        05F8  217 TEMPY      EQU    $5F8          ;used by scroll
C100:              218 *
C100:        047B  219 OLDCH      EQU    $478+3        ;last value of CH
C100:        057B  220 OURCH      EQU    $578+3        ;80-COL CH
C100:        05FB  221 OURCV      EQU    $5F8+3        ;CURSOR VERTICAL
C100:        067B  222 VFACTV     EQU    $678+3        ;Bit7=video firmware inactive
C100:        06FB  223 XCOORD     EQU    $6F8+3        ;X-COORD (GOTOXY)
C100:        077B  224 NXTCUR     EQU    $778+3        ;next cursor to display
C100:        07FB  225 CURSOR     EQU    $7F8+3        ;the current cursor char
C100:               17            INCLUDE SERIAL       ;Equates for serial code
```

```
C100:              ****************************************
C100:              *
C100:              * Apple Lolly communications driver
C100:              *
C100:              * By
C100:              * Rich Williams
C100:              * August 1983
C100:              * November 5 - j.r.huston
C100:              *
C100:              ****************************************
C100:              *
C100:              * Command codes
C100:              *
C100:              * Default command char is ctrl-A (^A)
C100:              *
C100:              *     ^AnnB: Set baud rate to nn
C100:              *     ^AnnD: Set data format bits to nn
C100:              *     ^AI:   Enable video echo
C100:              *     ^AK:   Disable CRLF
C100:              *     ^AL:   Enable CRLF
C100:              *     ^AnnN: Disable video echo & set printer width
C100:              *     ^AnnP: Set parity bits to nn
C100:              *     ^AQ    Quit terminal mode
C100:              *     ^AR    Reset the ACIA, IN#0 PR#0
C100:              *     ^AS    Send a 233 ms break character
C100:              *     ^AT    Enter terminal mode
C100:              *     ^AZ:   Zap control commands
C100:              *     ^Ax:   Set command char to ^x
C100:              *     ^AnnCR:Set printer width (CR = carriage return)
C100:              *
C100:              ****************************************
C100:        C100  serslot   equ   $C100
C100:        C200  comslot   equ   $C200
C100:                        MSB   ON
C100:        00BF  cmdcur    equ   '?'          ;Cursor while on command mode
C100:        00DF  termcur   equ   '_'          ;Cursor while in terminal mode
C100:                        MSB   OFF
C100:        0091  xon       equ   $91          ;XON character
C100:        03B8  sermode   equ   $3B8         ;D7=1 if in cmd; D6=1 if term 479 & 47A
C100:        0438  astat     equ   $438         ;Acia status from int 4F9 & 4FA
C100:        04B8  pwdth     equ   $4B8         ;Printer width 579 & 57A
C100:        0538  extint    equ   $538         ;extint & typhed enable 5F9 & 5FA
C100:        05F9  extint2   equ   $5F9
C100:        05FA  typhed    equ   $5FA
C100:        0679  oldcur    equ   $679         ;Saves cursor while in command
C100:        067A  oldcur2   equ   $67A         ;Saves cursor while in terminal mode
C100:        0638  eschar    equ   $638         ;Current escape character 6F9 & 6FA
C100:        06B8  flags     equ   $6B8         ;D7 = Video echo D6 = CRLF 779 & 77A
C100:        0738  col       equ   $738         ;Current printer column 7F9 & 7FA
C100:        047F  number    equ   $47F         ;Number accumulated in command
C100:        04FF  aciabuf   equ   $4FF         ;Owner of serial buffer
C100:        057F  twser     equ   $57F         ;Storage pointer for serial buffer
C100:        05FF  twkey     equ   $5FF         ;Storage pointer for type ahead buffer
C100:        067F  trser     equ   $67F         ;Retrieve pointer for serial buffer
C100:        06FF  trkey     equ   $6FF         ;Retrieve buffer for type ahead buffer
C100:        0800  thbuf     equ   $800         ;Buffer in alt ram space
C100:        06F8  temp      equ   $6F8         ;Temp storage
C100:        BFF8  sdata     equ   $BFF8        ;+$N0+$90 is output port
```

```
C100:         BFF9   61 sstat      equ    $BFF9          ;ACIA status register
C100:         BFFA   62 scomd      equ    $BFFA          ;ACIA command register
C100:         BFFB   63 scntl      equ    $BFFB          ;ACIA control register
C100:         FF58   64 iorts      equ    $FF58          ;RTS opcode
C100:                18            INCLUDE SER           ;Printer port @ $C100
```

```
C100:                *org serslot
C100:2C 58 FF                  bit   iorts        ;Set V to indicate initial entry
C103:70 0C    C111             bvs   entr1        ;Always taken
C105:38                        sec                 ;Input entry point
C106:90                        dfb   $90          ;BCC opcode
C107:18                        clc
C108:B8                        clv                 ;V = 0 since not initial entry
C109:50 06    C111             bvc   entr1        ;Always taken

C10B:01                        dfb   $01          ;pascal signiture byte
C10C:31                        dfb   $31          ;device signiture
C10D:E4                        dfb   >plinit
C10E:EE                        dfb   >plread
C10F:F6                        dfb   >plwrite
C110:FB                        dfb   >plstatus

C111:DA              entr1     phx                 ;Save the reg
C112:A2 C1                     ldx   #<serslot    ;X = Cn
C114:4C 33 C2                  jmp   setup        ;Set mslot, etc
C117:90 05    C11E   serport   bcc   serisout     ;Only output allowed
C119:20 4D CE                  jsr   zzquit       ;Reset the hooks
C11C:80 6A    C188             bra   done
C11E:0A              serisout  asl   A            ;A = flags
C11F:7A                        ply                 ;Get char
C120:5A                        phy
C121:BD B8 04                  lda   pwdth,x      ;Formatting enabled?
C124:F0 42    C168             beq   prnow
C126:A5 24                     lda   ch           ;Get current horiz position
C128:B0 1C    C146             bcs   servid       ;Branch if video echo
C12A:DD B8 04                  cmp   pwdth,x      ;If CH >= PWIDTH, then CH = COL
C12D:90 03    C132             bcc   chok
C12F:BD 38 07                  lda   col,x
C132:DD 38 07        chok      cmp   col,x        ;Must be > col for valid tab
C135:B0 0B    C142             bcs   fixch        ;Branch if ok
C137:C9 11                     cmp   #$11         ;8 or 16?
C139:B0 11    C14C             bcs   prnt         ;If > forget it
C13B:09 F0                     ora   #$F0         ;Find next comma cheaply
C13D:3D 38 07                  and   col,x        ;Don't blame me it's Dick's trick
C140:65 24                     adc   ch
C142:85 24           fixch     sta   ch           ;Save the new position
C144:80 06    C14C             bra   prnt
C146:C5 21           servid    cmp   wndwdth      ;If ch>= wndwdth go back to start of line
C148:90 02    C14C             blt   prnt
C14A:64 24                     stz   ch           ;Go back to left edge


C14C:                * We have a char to print
C14C:7A              prnt      ply
C14D:5A                        phy
C14E:BD 38 07                  lda   col,x        ;Have we exceeded width?
C151:DD B8 04                  cmp   pwdth,x
C154:B0 08    C15E             bge   toofar
C156:C5 24                     cmp   ch           ;Are we tabbing?
C158:B0 0E    C168             bge   prnow
C15A:A9 40                     lda   #$40         ;Space * 2
C15C:80 02    C160             bra   tab
C15E:A9 1A           toofar    lda   #$1A         ;CR * 2
```

```
C160:C0 80          tab      cpy   #$80         ;C = High bit
C162:6A                      ror   A            ;Shift it into char
C163:20 9D C1                jsr   serout3      ;Out it goes
C166:80 E4    C14C           bra   prnt
C168:98             prnow    tya
C169:20 8C C1                jsr   serout       ;Print the actual char
C16C:BD B8 04                lda   pwdth,x      ;Formatting enabled
C16F:F0 17    C188           beq   done
C171:3C B8 06                bit   flags,x      ;In video echo?
C174:30 12    C188           bmi   done
C176:BD 38 07                lda   col,x        ;Check if within 8 chars of right edge
C179:FD B8 04                sbc   pwdth,x      ;So BASIC can format output
C17C:C9 F8                   cmp   #$F8
C17E:90 04    C184           bcc   setch        ;If not within 8, we're done
C180:18                      clc
C181:65 21                   adc   wndwdth
C183:AC                      dfb   $AC          ;Dummy LDY to skip next two bytes
C184:A9 00          setch    lda   #0           ;Keep cursor at 0 if video off
C186:85 24                   sta   ch
C188:68             done     pla                 ;Restore regs
C189:7A                      ply
C18A:FA                      plx
C18B:60             socmd    rts



C18C:         C18C  serout   equ   *            ;Serial output
C18C:20 EB C9                jsr   command      ;Check if command
C18F:90 FA    C18B           bcc   socmd        ;All done if it is
C191:         C191  serout2  equ   *
C191:3C B8 06                bit   flags,x      ;N=1 iff video on
C194:10 07    C19D           bpl   serout3
C196:C9 91                   cmp   #xon         ;Don't echo ^Q
C198:F0 03    C19D           beq   serout3
C19A:20 F0 FD                jsr   coutl        ;Echo it
C19D:         C19D  serout3  equ   *
C19D:BC 85 C8                ldy   devno,x      ;Y points to ACIA
C1A0:48                      pha                 ;Save the char
C1A1:2C 58 FF                bit   iorts        ;Control char?
C1A4:F0 03    C1A9           beq   sordy        ;Don't inc column if so
C1A6:FE 38 07                inc   col,x
C1A9:08             sordy    php                ;can't have real interrupts for a while
C1AA:78                      sei
C1AB:B9 F9 BF                lda   sstat,y      ;Check XMIT empty & DCD
C1AE:10 11    C1C1           bpl   sordy2       ;branch if not clearing an interrupt
C1B0:48                      pha                ;save original status
C1B1:5A                      phy
C1B2:2C 14 C0                bit   rdramwrt     ;Save state of aux ram
C1B5:08                      php
C1B6:20 1C C9                jsr   aitst2
C1B9:28                      plp
C1BA:10 03    C1BF           bpl   somain       ;Branch if was main
C1BC:8D 05 C0                sta   wrcardram    ;Was alt ram
C1BF:7A             somain   ply
C1C0:68                      pla
C1C1:28             sordy2   plp
C1C2:29 30                   and   #$30
C1C4:C9 10                   cmp   #$10
```

```
C1C6:D0 E1    C1A9           bne   sordy
C1C8:68                      pla
C1C9:48                      pha                  ;Get char to XMIT
C1CA:99 F8 BF                sta   sdata,y       ;Out it goes
C1CD:3C B8 06                bit   flags,x       ;V=1 if LF after CR
C1D0:49 0D                   eor   #$0D          ;check for CR.
C1D2:0A                      asl   A             ;preserve bit 7
C1D3:D0 0D    C1E2           bne   sodone        ;branch if not CR.
C1D5:50 06    C1DD           bvc   clrcol        ;branch if no LF after CR
C1D7:A9 14                   lda   #$14          ;Get LF*2
C1D9:6A                      ror   A             ;no shift in high bit
C1DA:20 9D C1                jsr   serout3       ;Output the LF but don't echo it
C1DD:64 24         clrcol    stz   ch            ;0 position & column
C1DF:9E 38 07                stz   col,x
C1E2:68            sodone    pla                  ;Get the char back
C1E3:60                      rts
```

```
C1E4:                   * Pascal support stuff

C1E4:48                 plinit    pha
C1E5:20 C8 C2                     jsr    default        ;set defaults, enable acia
C1E8:9E B8 06                     stz    flags,x
C1EB:68                           pla
C1EC:80 05    C1F3                bra    plread2        ;all done...

C1EE:20 C5 C8           plread    jsr    XRDSER         ;read data from serial port (or buffer)
C1F1:90 FB    C1EE                bcc    plread         ;Branch if data not ready
C1F3:A2 00              plread2   ldx    #0
C1F5:60                           rts

C1F6:20 8C C1           plwrite   jsr    serout         ;Go output character
C1F9:80 F8    C1F3                bra    plread2
C1FB:80 1A    C217      plstatus  bra    p2status

C1FD:         0003                ds     comslot-*,$00
C200:                             INCLUDE COMM          ;Communications port @ $C200
```

```
C200:2C 58 FF                 bit   iorts         ;Set V to indicate initial entry
C203:70 2B    C230            bvs   entr
C205:38             sin       sec                  ;Input entry point
C206:90                       dfb   $90           ;BCC opcode to skip next byte
C207:18             sout      clc                  ;Output entry point
C208:B8                       clv                  ;Mark not initial entry
C209:50 25    C230            bvc   entr          ;Branch around pascal entry stuff

C20B:01                       dfb   $01           ;pascal signiture byte
C20C:31                       dfb   $31           ;device signiture
C20D:11                       dfb   >p2init
C20E:13                       dfb   >p2read
C20F:15                       dfb   >p2write
C210:17                       dfb   >p2status

C211:               * Pascal support stuff

C211:80 D1    C1E4  p2init    bra   p1init
C213:80 D9    C1EE  p2read    bra   p1read
C215:80 DF    C1F6  p2write   bra   p1write

C217:         C217  p2status  equ   *
C217:A2 40                    ldx   #$40          ;anticipate bad status request
C219:4A                       lsr   a             ;shift request to carry
C21A:D0 12    C22E            bne   notrdy
C21C:AA                       tax                 ;clear x for no error return code
C21D:A9 08                    lda   #8            ;anticpate input ready request
C21F:B0 01    C222            bcs   pstat2        ;branch if good guess.
C221:0A                       asl   a
C222:09 20          pstat2    ora   #$20          ;include DCD in test
C224:39 89 C0                 and   sstat+$90,y
C227:F0 05    C22E            beq   notrdy        ;branch if not ready for I/O
C229:49 20                    eor   #$20
C22B:38                       sec                 ;assume port is ready
C22C:D0 01    C22F            bne   isrdy         ;branch if good assumption
C22E:18             notrdy    clc                 ;indicate acia not ready for I/O
C22F:60             isrdy     rts

C230:DA             entr      phx
C231:A2 C2                    ldx   #<comslot     ;X = <CN00
C233:         C233  setup     equ   *
C233:5A                       phy
C234:48                       pha
C235:8E F8 07                 stx   mslot
C238:50 22    C25C            bvc   sudone        ;First call?
C23A:A5 36                    lda   cswl          ;If both hooks CN00 setup defaults
C23C:45 38                    eor   kswl
C23E:F0 06    C246            beq   sudodef
C240:A5 37                    lda   cswh          ;If both hooks CN then don't do def
C242:C5 39                    cmp   kswh          ;since it has already been done
C244:F0 03    C249            beq   sunodef
C246:20 C8 C2       sudodef   jsr   default       ;Set up defaults
C249:8A             sunodef   txa
C24A:45 39                    eor   kswh          ;Input call?
C24C:05 38                    ora   kswl
C24E:D0 07    C257            bne   suout         ;Must be Cn00
C250:A9 05                    lda   #>sin         ;Fix the input hook
C252:85 38                    sta   kswl
```

```
C254:38                         sec                  ;C = 1 for input call
C255:80 05    C25C              bra  sudone
C257:A9 07          suout       lda  #>sout         ;Fix output hook
C259:85 36                      sta  cswl           ;Note C might not be 0
C25B:18                         clc                  ;C=0 for output
C25C:BD B8 06       sudone      lda  flags,x        ;Check if serial or comm port
C25F:89 01                      bit  #1             ;Leave flags in a for serport
C261:D0 03    C266              bne  commport
C263:4C 17 C1       comout      jmp  serport
C266:90 FB    C263  commport    bcc  comout         ;Output?
C268:68                         pla                  ;Get the char
C269:80 28    C293              bra  terml          ;Input
C26B:3C B8 03       noesc       bit  sermode,x      ;In terminal mode?
C26E:50 1C    C28C              bvc  exitl          ;If not, return key
C270:20 91 C1                   jsr  serout2        ;Out it goes
C273:80 1E    C293              bra  terml
C275:         C275  testkbd     equ  *
C275:68                         pla                  ;Get current char
C276:20 70 CC                   jsr  update         ;Update cursor & check keyboard
C279:10 1B    C296              bpl  serin          ;N=0 if no new key
C27B:20 EB C9                   jsr  command        ;Test for command
C27E:B0 EB    C26B              bcs  noesc          ;Branch if not
C280:29 5F                      and  #$5f           ;upshift for following tests
C282:C9 51                      cmp  #'Q'           ;Quit?
C284:F0 04    C28A              beq  exitX
C286:C9 52                      cmp  #'R'           ;Reset?
C288:D0 09    C293              bne  terml          ;Go check serial
C28A:A9 98          exitX       lda  #$98           ;return a CTRL-X
C28C:7A             exitl       ply
C28D:FA                         plx
C28E:60                         rts
C28F:18             goremote    clc                  ;Into remote mode
C290:20 CD CA       goterm      jsr  setterm        ;Into terminal mode
C293:         C293  terml       equ  *
C293:20 4C CC                   jsr  showcur        ;Get current char on screen
C296:48             serin       pha
C297:20 C5 C8                   jsr  XRDSER         ;Is it ready?
C29A:90 D9    C275              bcc  testkbd        ;If not, try the keyboard
C29C:A8                         tay                  ;Save new input in y for now
C29D:68                         pla
C29E:5A                         phy                  ;Save new char on stack
C29F:20 B8 C3                   jsr  storch         ;Fix the screen
C2A2:68                         pla                  ;Get the new data
C2A3:BC 38 06                   ldy  eschar,x       ;If 0, don't modify char
C2A6:F0 16    C2BE              beq  sinomod
C2A8:09 80                      ora  #$80           ;Apple loves the high bit
C2AA:C9 8A                      cmp  #$8A           ;Ignore line feed
C2AC:F0 E5    C293              beq  terml
C2AE:C9 91                      cmp  #xon
C2B0:F0 E1    C293              beq  terml          ;Ignore ^Q
C2B2:C9 FF                      cmp  #$FF           ;Ignore FFs
C2B4:F0 DD    C293              beq  terml
C2B6:C9 92                      cmp  #$92           ;^R for remote?
C2B8:F0 D5    C28F              beq  goremote
C2BA:C9 94                      cmp  #$94           ;^T for terminal mode?
C2BC:F0 D2    C290              beq  goterm
C2BE:3C B8 03       sinomod     bit  sermode,x      ;In terminal mode?
C2C1:50 C9    C28C              bvc  exitl          ;Return to user if not A = char
```

```
C2C3:20 ED FD                   jsr   cout             ;Onto the screen with it
C2C6:80 CB    C293              bra   terml
C2C8:         C2C8    default   equ   *                ;Set up the defaults
C2C8:20 A2 C8                   jsr   moveirq          ;make sure irq vectors ok
C2CB:BC 3B C2                   ldy   defidx-$C1,x     ;Index into alt screen. Table in command
C2CE:20 7C C3         defloop   jsr   getalt           ;Get default from alt screen
C2D1:48                         pha
C2D2:88                         dey
C2D3:30 04    C2D9              bmi   defff            ;Done if minus
C2D5:C0 03                      cpy   #3
C2D7:D0 F5    C2CE              bne   defloop          ;Or if 2
C2D9:20 A2 C8         defff     jsr   moveirq          ;Jam irq vector into LC
C2DC:68                         pla                     ;Command, control & flags on stack
C2DD:BC 85 C8                   ldy   devno,x
C2E0:99 FB BF                   sta   scntl,y          ;Set command reg
C2E3:68                         pla
C2E4:99 FA BF                   sta   scomd,y
C2E7:68                         pla
C2E8:9D B8 06                   sta   flags,x          ;And the flags
C2EB:29 01                      and   #1               ;A = $01 (^A) if comm mode
C2ED:D0 02    C2F1              bne   defcom
C2EF:A9 09                      lda   #9               ;^I for serial port
C2F1:9D 38 06         defcom    sta   eschar,x
C2F4:68                         pla                     ;Get printer width
C2F5:9D B8 04                   sta   pwdth,x
C2F8:9E B8 03                   stz   sermode,x
C2FB:60                         rts
C2FC:03 07            defidx    dfb   3,7
C2FE:         0002              ds    $C300-*,$00
C300:                           INCLUDE C3SPACE        ;80 column card @ $C300
```

```
C300:               ****************************************
C300:               *
C300:               * THIS IS THE $C3XX ROM SPACE:
C300:               *
C300:               ****************************************
C300:48             C3ENTRY   PHA                 ;save regs
C301:DA                       PHX
C302:5A                       PHY
C303:80 12    C317            BRA   BASICINIT     ;and init video firmware
C305:38             C3KEYIN   SEC                 ;Pascal 1.1 ID byte
C306:90                       DFB   $90           ;BCC OPCODE (NEVER TAKEN)
C307:18             C3COUT1   CLC                 ;Pascal 1.1 ID byte
C308:80 1A    C324            BRA   BASICENT      ;=>go print/read char
C30A:EA                       NOP
C30B:               *
C30B:               * PASCAL 1.1 FIRMWARE PROTOCOL TABLE:
C30B:               *
C30B:01                       DFB   $01           ;GENERIC SIGNATURE BYTE
C30C:88                       DFB   $88           ;DEVICE SIGNATURE BYTE
C30D:               *
C30D:2C                       DFB   >JPINIT       ;PASCAL INIT
C30E:2F                       DFB   >JPREAD       ;PASCAL READ
C30F:32                       DFB   >JPWRITE      ;PASCAL WRITE
C310:35                       DFB   >JPSTAT       ;PASCAL STATUS
C311:               ****************************************
C311:               *
C311:               * 128K SUPPORT ROUTINE ENTRIES:
C311:               *
C311:4C 86 CF                 JMP   MOVEAUX       ;MEMORY MOVE ACROSS BANKS
C314:4C CD CF                 JMP   XFER          ;TRANSFER ACROSS BANKS
C317:               ****************************************
C317:               *
C317:               ****************************************
C317:               * BASIC I/O ENTRY POINT:
C317:               ****************************************
C317:               *
C317:20 20 CE       BASICINIT JSR   HOOKUP        ;COPYROM if needed, sethooks
C31A:20 BE CD                 JSR   SET80         ;setup 80 columns
C31D:20 58 FC                 JSR   HOME          ;clear screen
C320:7A                       PLY
C321:FA                       PLX                 ;restore X
C322:68                       PLA                 ;restore char
C323:18                       CLC                 ;output a character
C324:               *
C324:B0 03    C329  BASICENT  BCS   BINPUT        ;=>carry me to input
C326:4C F6 FD       BPRINT    JMP   COUTZ         ;print a character
C329:4C 1B FD       BINPUT    JMP   KEYIN         ;get a keystroke
C32C:               *
C32C:4C 41 CF       JPINIT    JMP   PINIT         ;pascal init
C32F:4C 35 CF       JPREAD    JMP   PASREAD       ;pascal read
C332:4C C2 CE       JPWRITE   JMP   PWRITE        ;pascal write
C335:4C B1 CE       JPSTAT    JMP   PSTATUS       ;pascal status call
C338:               *
C338:               * COPYROM is called when the video firmware is
C338:               * initialized.  If the language card is switched
C338:               * in for reading, it copies the F8 ROM to the
C338:               * language card and restores the state of the
C338:               * language card.
```

```
C338:                *
C338:A9 06           COPYROM   LDA   #GOODF8       ;get the ID byte
C33A:                *
C33A:                * Compare ID bytes to whatever is readable.  If it
C33A:                * matches, all is ok.  If not, need to copy.
C33A:                *
C33A:CD B3 FB                  CMP   F8VERSION     ;does it match?
C33D:F0 3C    C37B             BEQ   ROMOK
C33F:20 60 C3                  JSR   SETROM        ;read ROM, write RAM, save state
C342:A9 F8                     LDA   #$F8          ;from F800-FFFF
C344:85 37                     STA   CSWH
C346:64 36                     STZ   CSWL
C348:B2 36           COPYROM2  LDA   (CSWL)        ;get a byte
C34A:92 36                     STA   (CSWL)        ;and save a byte
C34C:E6 36                     INC   CSWL
C34E:D0 F8    C348             BNE   COPYROM2
C350:E6 37                     INC   CSWH
C352:D0 F4    C348             BNE   COPYROM2      ;fall into RESETLC
C354:                *
C354:                * RESETLC resets the language card to the state
C354:                * determined by SETROM.  It always leaves the card
C354:                * write enabled.
C354:                *
C354:DA              RESETLC   PHX                 ;save X
C355:AE 78 04                  LDX   ROMSTATE      ;get the state
C358:3C 81 C0                  BIT   ROMIN,X       ;set bank & ROM/RAM read
C35B:3C 81 C0                  BIT   ROMIN,X       ;set write enable
C35E:FA                        PLX                 ;restore X
C35F:60                        RTS
C360:                *
C360:                * SETROM switches in the ROM for reading, the RAM
C360:                * for writing, and it saves the state of the
C360:                * language card.  It does not save the write
C360:                * protect status of the card.
C360:                *
C360:DA              SETROM    PHX                 ;save x
C361:A2 00                     LDX   #0            ;assume write enable,bank2,ROMRD
C363:2C 11 C0                  BIT   RDLCBNK2      ;is bank 2 switched in?
C366:30 02    C36A             BMI   NOT1          ;=>yes
C368:A2 08                     LDX   #$8           ;indicate bank 1
C36A:2C 12 C0        NOT1      BIT   RDLCRAM       ;is LC RAM readable?
C36D:10 02    C371             BPL   NOREAD        ;=>no
C36F:E8                        INX                 ;indicate RAM read
C370:E8                        INX
C371:2C 81 C0        NOREAD    BIT   $C081         ;ROM read
C374:2C 81 C0                  BIT   $C081         ;RAM write
C377:8E 78 04                  STX   ROMSTATE      ;save state
C37A:FA                        PLX                 ;restore X
C37B:60              ROMOK     RTS
C37C:                *
C37C:                * GETALT reads a byte from aux memory screenholes.
C37C:                * Y is the index to the byte (0-7) indexed off of
C37C:                * address $478.
C37C:                *
C37C:AD 13 C0        GETALT    LDA   RDRAMRD       ;save state of aux memory
C37F:0A                        ASL   A
C380:AD 18 C0                  LDA   RD80COL       ;and of the 80STORE switch
C383:08                        PHP
```

```
C384:8D 00 C0                 STA   CLR80COL      ;no 80STORE to get page 1
C387:8D 03 C0                 STA   RDCARDRAM     ;pop in the other half of RAM
C38A:B9 78 04                 LDA   $478,Y        ;read the desired byte
C38D:28                       PLP                 ;and restore memory
C38E:B0 03   C393             BCS   GETALT1
C390:8D 02 C0                 STA   RDMAINRAM
C393:10 03   C398  GETALT1    BPL   GETALT2
C395:8D 01 C0                 STA   SET80COL
C398:60            GETALT2    RTS
C399:              *
C399:09 80         UPSHIFT0   ORA   #$80          ;set high bit for execs
C39B:C9 FB         UPSHIFT    CMP   #$FB
C39D:B0 06   C3A5             BCS   X.UPSHIFT
C39F:C9 E1                    CMP   #$E1
C3A1:90 02   C3A5             BCC   X.UPSHIFT
C3A3:29 DF                    AND   #$DF
C3A5:60            X.UPSHIFT RTS
C3A6:              *
C3A6:              * GETCOUT performs COUT for GETLN.  It disables the
C3A6:              * echoing of control characters by clearing the
C3A6:              * M.CTL mode bit, prints the char, then restores
C3A6:              * M.CTL.  NOESC is used by the RDKEY routine to
C3A6:              * disable escape sequences.
C3A6:              *
C3A6:48            GETCOUT    PHA                 ;save char to print
C3A7:A9 08                    LDA   #M.CTL        ;disable control chars
C3A9:1C FB 04                 TRB   VMODE         ;by clearing M.CTL
C3AC:68                       PLA                 ;restore character
C3AD:20 ED FD                 JSR   COUT          ;and print it
C3B0:4C 44 FD                 JMP   NOESCAPE      ;enable control chars
C3B3:              *
C3B3:              * STORCH determines loads the current cursor position,
C3B3:              *         inverts the character, and displays it
C3B3:              * STORCHAR inverts the character and displays it at the
C3B3:              *         position stored in Y
C3B3:              * STORY  determines the current cursor position, and
C3B3:              *         displays the character without inverting it
C3B3:              * STORE  displays the char at the position in Y
C3B3:              *
C3B3:              * If mouse characters are enabled (VMODE bit 0 = 0)
C3B3:              * then mouse characters ($40-$5F) are displayed when
C3B3:              * the alternate character set is switched in.  Normally
C3B3:              * values $40-$5F are shifted to $0-$1F before display.
C3B3:              *
C3B3:              * Calls to GETCUR trash Y
C3B3:              *
C3B3:20 9D CC      STORY      JSR   GETCUR        ;get newest cursor into Y
C3B6:80 09   C3C1             BRA   STORE
C3B8:              *
C3B8:20 9D CC      STORCH     JSR   GETCUR        ;first, get cursor position
C3BB:24 32                    BIT   INVFLG        ;normal or inverse?
C3BD:30 02   C3C1             BMI   STORE         ;=>normal, store it
C3BF:29 7F                    AND   #$7F          ;inverse it
C3C1:5A            STORE      PHY                 ;save real Y
C3C2:09 00                    ORA   #0            ;does char have high bit set?
C3C4:30 15   C3DB             BMI   STORE1        ;=>yes, don't do mouse check
C3C6:48                       PHA                 ;save char
C3C7:AD FB 04                 LDA   VMODE         ;is mouse bit set?
```

```
C3CA:6A                        ROR   A
C3CB:68                        PLA                  ;restore char
C3CC:90 0D    C3DB             BCC   STORE1         ;=>no, don't do mouse shift
C3CE:2C 1E C0                  BIT   ALTCHARSET     ;no shift if ][ char set
C3D1:10 08    C3DB             BPL   STORE1         ;=> it is!
C3D3:49 40                     EOR   #$40           ;$40-$5F=>0-$1f
C3D5:89 60                     BIT   #$60
C3D7:F0 02    C3DB             BEQ   STORE1
C3D9:49 40                     EOR   #$40
C3DB:2C 1F C0        STORE1    BIT   RD80VID        ;80 columns?
C3DE:10 19    C3F9             BPL   STORE5         ;=>no, store char
C3E0:48                        PHA                  ;save (shifted) char
C3E1:8D 01 C0                  STA   SET80COL       ;hit 80 store
C3E4:98                        TYA                  ;get proper Y
C3E5:45 20                     EOR   WNDLFT         C=1 if char in main ram
C3E7:4A                        LSR   A
C3E8:B0 04    C3EE             BCS   STORE2         ;=>yes, main RAM
C3EA:AD 55 C0                  LDA   TXTPAGE2       ;else flip in aux RAM
C3ED:C8                        INY                  ;do this for odd left, aux bytes
C3EE:98              STORE2    TYA                  ;divide pos'n by 2
C3EF:4A                        LSR   A
C3F0:A8                        TAY
C3F1:68                        PLA                  ;get (shifted) char
C3F2:91 28           STORE3    STA   (BASL),Y       ;stuff it
C3F4:2C 54 C0                  BIT   TXTPAGE1       ;else restore page1
C3F7:7A              STORE4    PLY                  ;restore real Y
C3F8:60                        RTS                  ;und exit
C3F9:                *
C3F9:91 28           STORE5    STA   (BASL),Y       ;do 40 column store
C3FB:7A                        PLY                  ;restore Y
C3FC:60                        RTS                  ;and exit
C3FD:         0003             DS    $C400-*,$00
C400:                          INCLUDE MOUSE        ;Equates for the mouse
```

```
C400:                        MSB   ON
C400:             *****************************************
C400:             *
C400:             * Mouse firmware for the Chels
C400:             *
C400:             *  by Rich Williams
C400:             *  July, 1983
C400:             *
C400:             *****************************************


C400:             *****************************************
C400:             *
C400:             * Equates
C400:             *
C400:             *****************************************


C400:             * Input bounds are in scratch area
C400:       0478  moutemp   equ   $478          ;Temporary storage
C400:       0478  minl      equ   $478
C400:       04F8  maxl      equ   $4F8
C400:       0578  minh      equ   $578
C400:       05F8  maxh      equ   $5F8
C400:             * Mouse bounds in slot 5 screen area
C400:       047D  minxl     equ   $47D
C400:       04FD  minyl     equ   $4FD
C400:       057D  minxh     equ   $57D
C400:       05FD  minyh     equ   $5FD
C400:       067D  maxxl     equ   $67D
C400:       06FD  maxyl     equ   $6FD
C400:       077D  maxxh     equ   $77D
C400:       07FD  maxyh     equ   $7FD
C400:             * Mouse holes in slot 4 screen area
C400:       047C  mouxl     equ   $47C          ;X position low byte
C400:       04FC  mouyl     equ   $4FC          ;Y position low byte
C400:       057C  mouxh     equ   $57C          ;X position high byte
C400:       05FC  mouyh     equ   $5FC          ;Y position high byte
C400:       067C  mouarm    equ   $67C          ;Arm interrupts from movement or button
C400:       077C  moustat   equ   $77C          ;Mouse status
C400:             * Moustat provides the following
C400:             *  D7= Button pressed
C400:             *  D6= Status of button on last read
C400:             *  D5= Moved since last read
C400:             *  D4= Reserved
C400:             *  D3= Interrupt from VBL
C400:             *  D2= Interrupt from button
C400:             *  D1= Interrupt from movement
C400:             *  D0= Reserved
C400:       07FC  moumode   equ   $7FC          ;Mouse mode
C400:             *  D7-D4= Unused
C400:             *  D3= VBL active
C400:             *  D2= VBL interrupt on button
C400:             *  D1= VBL interrupt on movement
C400:             *  D0= Mouse active
C400:       0020  movarm    equ   $20
```

```
C400:        000C     vblmode    equ    $0C
C400:        0004     butmode    equ    $04              ;D2 mask
C400:        0002     movmode    equ    $02              ;D1 mask


C400:                 * Hardware addresses
C400:        C015     mouxint    equ    $C015            ;D7 = x interrupt
C400:        C017     mouyint    equ    $C017            ;D7 = y interrupt
C400:        C019     vblint     equ    $C019            ;D7 = vbl interrupt
C400:        C078     ioudsbl    equ    $C078            ;Disable iou access
C400:        C079     iouenbl    equ    $C079            ;Enable iou access
C400:        C048     mouclr     equ    $C048            ;Clear mouse interrupt
C400:        C058     iou        equ    $C058            ;IOU interrupt switches
C400:        C058     moudsbl    equ    $C058            ;Disable mouse interrupts
C400:        C059     mouenbl    equ    $C059            ;Enable mouse interrupts
C400:        C063     moubut     equ    $C063            ;D7 = Mouse button
C400:        C066     mouxl      equ    $C066            ;D7 = X1
C400:        C067     mouyl      equ    $C067            ;D7 = Y1
C400:        C070     vblclr     equ    $C070            ;Clear VBL interrupt
C400:                 *
C400:                 * Other addresses
C400:                 *
C400:        0200     inbuf      equ    $200             ;Input buffer
C400:        0214     binl       equ    inbuf+20         ;Temp for binary conversion
C400:        0215     binh       equ    inbuf+21
C400:                                   INCLUDE MCODE    ;Mouse @ $C400
```

```
C400:                  ******************************************
C400:                  *
C400:                  * Entry points for mouse firmware
C400:                  *
C400:                  ******************************************
C400:80 05    C407     mbasic     bra    outent
C402:A2 03             pnull      ldx    #3
C404:60                           rts                   ;Null for pascal entry
C405:38                inent      sec                   ;Signature bytes
C406:90                           dfb    $90
C407:18                outent     clc
C408:4C 80 C7                     jmp    xmbasic       ;Go do basic entry
C40B:01                           dfb    $01           ;More signature stuff
C40C:20                           dfb    $20
C40D:02                           dfb    >pnull
C40E:02                           dfb    >pnull
C40F:02                           dfb    >pnull
C410:02                           dfb    >pnull
C411:00                           dfb    $0
C412:3D                           dfb    >xsetmou      ;SETMOUSE
C413:FC                           dfb    >xmtstint     ;SERVEMOUSE
C414:95                           dfb    >xmread       ;READMOUSE
C415:84                           dfb    >xmclear      ;CLEARMOUSE
C416:6B                           dfb    >noerror      ;POSMOUSE
C417:B0                           dfb    >xmclamp      ;CLAMPMOUSE
C418:6D                           dfb    >xmhome       ;HOMEMOUSE
C419:1C                           dfb    >initmouse    ;INITMOUSE
C41A:02                           dfb    >pnull
C41B:CF                           dfb    >xmint
```

```
C41C:                 ****************************************
C41C:                 *
C41C:                 * Initmouse - resets the mouse
C41C:                 *  Also clears all of the mouse holes
C41C:                 * note that iou access fires pdlstrb & makes mouse happy
C41C:                 *
C41C:                 ****************************************
C41C:           C41C  initmouse equ    *
C41C:9C 7C 07                    stz   moustat       ;Clear status
C41F:A2 80                       ldx   #$80
C421:A0 01                       ldy   #1
C423:9E 7D 04         xrloop     stz   minxl,x       ;Minimum = $0000
C426:9E 7D 05                    stz   minxh,x
C429:A9 FF                       lda   #$FF          ;Maximum = $03FF
C42B:9D 7D 06                    sta   maxxl,x
C42E:A9 03                       lda   #03
C430:9D 7D 07                    sta   maxxh,x
C433:A2 00                       ldx   #0
C435:88                          dey
C436:10 EB      C423             bpl   xrloop
C438:20 6D C4                    jsr   xmhome        ;Clear the mouse holes
C43B:A9 00                       lda   #0            ;Fall into SETMOU


C43D:                 ****************************************
C43D:                 *
C43D:                 * XSETMOU - Sets the mouse mode to A
C43D:                 *
C43D:                 ****************************************
C43D:           C43D  xsetmou    equ   *
C43D:AA                          tax
C43E:20 A2 C8                    jsr   moveirq       ;Make sure interrupt vector is right
C441:8A                          txa                  ;Only x preserved by moveirq
C442:8D 78 04                    sta   moutemp
C445:4A                          lsr   A             ;D0 = 1 if mouse active
C446:0D 78 04                    ora   moutemp       ;D2 = 1 if vbl active
C449:C9 10                       cmp   #$10          ;If >=$10 then invalid mode
C44B:B0 1F      C46C             bcs   sminvalid
C44D:29 05                       and   #5            ;Extract VBL & Mouse
C44F:F0 01      C452             beq   xsoff         ;Turning it off?
C451:58                          cli                  ;If not, ints active
C452:69 55            xsoff      adc   #$55          ;Make iou byte C=0


C454:                 ****************************************
C454:                 *
C454:                 * SETIOU - Sets the IOU interrupt modes to A
C454:                 *  Inputs: A = Bits to change
C454:                 *  D7 = Y int on falling edge
C454:                 *  D6 = Y int on rising edge
C454:                 *  D5 = X int on falling edge
C454:                 *  D4 = X int on rising edge
C454:                 *  D3 = Enable VBL int
C454:                 *  D2 = Disable VBL int
C454:                 *  D1 = Enable mouse int
C454:                 *  D0 = Disable mouse int
```

```
C454:                *
C454:                *
C454:                *****************************************
C454:          C454  setiou    equ   *
C454:08                        php
C455:78                        sei                    ;Don't allow ints while iou enabled
C456:8E FC 07                  stx   moumode
C459:8D 79 C0                  sta   iouenbl         ;Enable iou access
C45C:A2 08                     ldx   #8
C45E:CA              siloop    dex
C45F:0A                        asl   A               ;Get a bit to check
C460:90 03    C465             bcc   sinoch          ;No change if C=0
C462:9D 58 C0                  sta   iou,x           ;Set it
C465:D0 F7    C45E   sinoch    bne   siloop          ;Any bits left in A?
C467:8D 78 C0                  sta   ioudsbl         ;Turn off iou access
C46A:28                        plp
C46B:18              noerror   clc
C46C:60              sminvalid rts
```

```
C46D:                *****************************************
C46D:                *
C46D:                * XMHOME- Clears mouse position & status
C46D:                *
C46D:                *****************************************
C46D:          C46D  xmhome    equ   *
C46D:A2 80                     ldx   #$80            ;Point mouse to upper left
C46F:80 02    C473             bra   xmh2
C471:A2 00           xmhloop   ldx   #0
C473:BD 7D 04        xmh2      lda   minxl,x
C476:9D 7C 04                  sta   mouxl,x
C479:BD 7D 05                  lda   minxh,x
C47C:9D 7C 05                  sta   mouxh,x
C47F:CA                        dex
C480:10 EF    C471             bpl   xmhloop
C482:80 0C    C490             bra   xmcdone
```

```
C484:                *****************************************
C484:                *
C484:                * XMCLEAR - Sets the mouse to 0,0
C484:                *
C484:                *****************************************
C484:          C484  xmclear   equ   *
C484:9C 7C 04                  stz   mouxl
C487:9C 7C 05                  stz   mouxh
C48A:9C FC 04                  stz   mouyl
C48D:9C FC 05                  stz   mouyh
C490:9C 7C 06        xmcdone   stz   mouarm
C493:18                        clc
C494:60                        rts
```

```
C495:                *****************************************
C495:                *
C495:                * XMREAD - Updates the screen holes
C495:                *
C495:                *****************************************
C495:          C495  xmread    equ   *
C495:A9 20                     lda   #movarm       ;Has mouse moved?
C497:2D 7C 06                  and   mouarm
C49A:1C 7C 06                  trb   mouarm        ;Clear arm bit
C49D:2C 63 C0                  bit   moubut        ;Button pressed?
C4A0:30 02     C4A4            bmi   xrbut
C4A2:09 80                     ora   #$80
C4A4:2C 7C 07        xrbut     bit   moustat       ;Pressed last time?
C4A7:10 02     C4AB            bpl   xrbut2
C4A9:09 40                     ora   #$40
C4AB:8D 7C 07        xrbut2    sta   moustat
C4AE:18                        clc
C4AF:60                        rts


C4B0:                *****************************************
C4B0:                *
C4B0:                * XMCLAMP - Store new bounds
C4B0:                *  Inputs A = 1 for Y, 0 for X axis
C4B0:                *    minl, minh, maxl, maxh = new bounds
C4B0:                *
C4B0:                *****************************************
C4B0:          C4B0  xmclamp   equ   *
C4B0:6A                        ror   A             ;1 -> 80
C4B1:6A                        ror   A
C4B2:29 80                     and   #$80
C4B4:AA                        tax
C4B5:AD 78 04                  lda   minl
C4B8:9D 7D 04                  sta   minxl,x
C4BB:AD 78 05                  lda   minh
C4BE:9D 7D 05                  sta   minxh,x
C4C1:AD F8 04                  lda   maxl
C4C4:9D 7D 06                  sta   maxxl,x
C4C7:AD F8 05                  lda   maxh
C4CA:9D 7D 07                  sta   maxxh,x
C4CD:18                        clc                  ;No error
C4CE:60                        rts
```

```
C4CF:               ****************************************
C4CF:               *
C4CF:               * Mouse interrupt handler
C4CF:               *
C4CF:               * MOUSEINT - Monitor's interrupt handler
C4CF:               * XMINT - Interrupt handler the user can use
C4CF:               * XMTSTINT - Checks mouse status bits
C4CF:               ****************************************
C4CF:          C4CF xmint     equ   *
C4CF:AE 66 C0                 ldx   mouxl          ;Get Xl & Yl asap
C4D2:AC 67 C0                 ldy   mouyl
C4D5:          C4D5 mouseint  equ   *              ;Entry point if X & Y set up
C4D5:A9 0E                    lda   #$0E           ;Clear status bits
C4D7:1C 7C 07                 trb   moustat



C4DA:38                       sec                   ;Assume interrupt not handled
C4DB:               * Check for vertical blanking interrupt
C4DB:AD 19 C0                 lda   vblint         ;VBL interrupt?
C4DE:10 48     C528           bpl   chkmou
C4E0:8D 79 C0                 sta   iouenbl        ;Enable iou access & clear VBL interrupt
C4E3:A9 0C                    lda   #vblmode       ;Should we leave vbl active?
C4E5:2C FC 07                 bit   moumode
C4E8:D0 03     C4ED           bne   cvnovbl
C4EA:8D 5A C0                 sta   iou+2          ;Disable VBL
C4ED:09 02          cvnovbl   ora   #movmode
C4EF:80 1B     C50C           bra   xmskip



C4F1:A9 0E          mistat    lda   #$0E
C4F3:2D 7C 07                 and   moustat
C4F6:D0 01     C4F9           bne   nostat2
C4F8:38                       sec
C4F9:68             nostat2   pla
C4FA:60                       rts
C4FB:          0000           ds    $C4FB-*
C4FB:D6                       dfb   $D6            ;Signature byte
C4FC:48             xmtstint  pha
C4FD:18                       clc
C4FE:80 F1     C4F1           bra   mistat         ;Go check status
C500:FF                       dfb   $FF
C501:20 4D CE                 jsr   zzquit         ;Get out of the hooks
C504:A2 FF                    ldx   #$FF
C506:20 24 CB       qloop     jsr   zznm2
C509:10 FB     C506           bpl   qloop
C50B:60                       rts
C50C:          C50C xmskip    equ   *
C50C:8D 78 C0                 sta   ioudsbl
C50F:2C 7C 06                 bit   mouarm         ;VBL bit in arm isn't used
C512:D0 02     C516           bne   cvmoved
C514:A9 0C                    lda   #vblmode       ;Didn't move
C516:2C 63 C0       cvmoved   bit   moubut         ;Button pressed?
C519:10 02     C51D           bpl   cvbut
C51B:49 04                    eor   #butmode       ;Clear the button bit
C51D:2D FC 07       cvbut     and   moumode        ;Which bits were set in the mode
C520:0C 7C 07                 tsb   moustat
```

```
C523:1C 7C 06                trb   mouarm
C526:69 FE                   adc   #$FE           ;C=1 if int passes to user
C528:              * Check & update mouse movement
C528:         C528 chkmou    equ   *
C528:AD 15 C0                lda   mouxint        ;Mouse interrupt?
C52B:0D 17 C0                ora   mouyint
C52E:10 6A    C59A           bpl   xmdone         ;If not return with C from vbl
C530:8A                      txa                   ;Get X1 in A
C531:A2 00                   ldx   #0
C533:2C 15 C0                bit   mouxint        ;X movement?
C536:30 0A    C542           bmi   cmxmov
C538:98            cmloop    tya                   ;Get Y1 into A
C539:49 80                   eor   #$80           ;Complement direction
C53B:A2 80                   ldx   #$80
C53D:2C 17 C0                bit   mouyint
C540:10 39    C57B           bpl   cmnoy
C542:0A            cmxmov    asl   A
C543:BD 7C 04                lda   mouxl,x        ;A = current low byte
C546:B0 1A    C562           bcs   cmrght         ;Which way?
C548:DD 7D 04                cmp   minxl,x        ;Move left
C54B:D0 08    C555           bne   cmlok
C54D:BD 7C 05                lda   mouxh,x
C550:DD 7D 05                cmp   minxh,x
C553:F0 22    C577           beq   cmnoint
C555:BD 7C 04      cmlok     lda   mouxl,x
C558:D0 03    C55D           bne   cmnt0          ;Borrow from high byte?
C55A:DE 7C 05                dec   mouxh,x
C55D:DE 7C 04      cmnt0     dec   mouxl,x
C560:80 15    C577           bra   cmnoint
C562:DD 7D 06      cmrght    cmp   maxxl,x        ;At high bound?
C565:D0 08    C56F           bne   cmrok
C567:BD 7C 05                lda   mouxh,x
C56A:DD 7D 07                cmp   maxxh,x
C56D:F0 08    C577           beq   cmnoint
C56F:FE 7C 04      cmrok     inc   mouxl,x        ;Move right
C572:D0 03    C577           bne   cmnoint
C574:FE 7C 05                inc   mouxh,x
C577:E0 00         cmnoint   cpx   #0
C579:F0 BD    C538           beq   cmloop
C57B:8D 48 C0      cmnoy     sta   mouclr
C57E:A9 02                   lda   #movmode       ;Should we enable VBL?
C580:2D FC 07                and   moumode
C583:F0 09    C58E           beq   cmnovbl        ;Branch if not
C585:8D 79 C0                sta   iouenbl
C588:8D 5B C0                sta   iou+3          ;Enable VBL int
C58B:8D 78 C0                sta   ioudsbl
C58E:09 20         cmnovbl   ora   #movarm        ;Mark that we moved
C590:0C 7C 06                tsb   mouarm
C593:A9 0E                   lda   #$0E
C595:2D 7C 07                and   moustat
C598:69 FE                   adc   #$FE           ;C=1 iff any bits were 1
C59A:60            xmdone    rts
```

```
C59B:                *****************************************
C59B:                *
C59B:                * HEXTODEC - Puts +0000, into the input buffer
C59B:                *  inputs: A = Low byte of number
C59B:                *          X = High byte of number
C59B:                *          Y = Position of ones digit
C59B:                *
C59B:                *****************************************
C59B:          C59B  hextodec equ   *
C59B:E0 80           cpx   #$80          ;Is it a negative number?
C59D:90 0D     C5AC           bcc   hexdec2
C59F:49 FF                    eor   #$FF          ;Form two's complement
C5A1:69 00                    adc   #0            ;C = 1 from compare
C5A3:48                       pha                  ;Save it
C5A4:8A                       txa
C5A5:49 FF                    eor   #$FF
C5A7:69 00                    adc   #0
C5A9:AA                       tax
C5AA:68                       pla
C5AB:38                       sec
C5AC:8D 14 02        hexdec2  sta   binl          ;Store the number to convert
C5AF:8E 15 02                 stx   binh
C5B2:A9 AB                    lda   #'+'          ;Store the sigh in the buffer
C5B4:90 02     C5B8           bcc   hdpos2
C5B6:A9 AD                    lda   #'-'
C5B8:48              hdpos2   pha                  ;Save the sign
C5B9:A9 AC                    lda   #','          ;Store a comma after the number
C5BB:99 01 02                 sta   inbuf+1,y
C5BE:          C5BE  hdloop   equ   *             ;Divide by 10
C5BE:                *
C5BE:                * Divide BINH,L by 10 and leave remainder in A
C5BE:                *
C5BE:A2 11                    ldx   #16+1         ;16 bits and first time do nothing
C5C0:A9 00                    lda   #0
C5C2:18                       clc                  ;C=0 so first ROL leaves A=0
C5C3:2A              dv10loop rol   A
C5C4:C9 0A                    cmp   #10           ;A >= 10?
C5C6:90 02     C5CA           bcc   dv10lt        ;Branch if <
C5C8:E9 0A                    sbc   #10           ;C = 1 from compare and is left set
C5CA:2E 14 02        dv10lt   rol   binl
C5CD:2E 15 02                 rol   binh
C5D0:CA                       dex
C5D1:D0 F0     C5C3           bne   dv10loop
C5D3:09 B0                    ora   #'0'          ;Make a ascii char
C5D5:99 00 02                 sta   inbuf,y
C5D8:88                       dey
C5D9:F0 08     C5E3           beq   hddone        ;Stop on 0,6,12
C5DB:C0 07                    cpy   #7
C5DD:F0 04     C5E3           beq   hddone
C5DF:C0 0E                    cpy   #14
C5E1:D0 DB     C5BE           bne   hdloop
C5E3:68              hddone   pla                  ;Get the sign
C5E4:99 00 02                 sta   inbuf,y
C5E7:60                       rts
C5E8:DF 67 37 1C     qtbl     dfb   $DF,$67,$37,$1C,$07,$0C,$45,$62
C5F0:6E 7E 3B 0A              dfb   $6E,$7E,$3B,$0A,$0B,$48,$77,$7B
C5F8:66 2B 0C 08              dfb   $66,$2B,$0C,$08,$16,$53,$68,$C5
C600:          0000           ds    $C600-*
```

```
C600:          0356   DNIBL      EQU   $356
C600:          0300   NBUF1      EQU   $300
C600:          07DB   BOOTSCRN   EQU   $7DB
C600:          002B   SLOTZ      EQU   $2B
C600:          003C   BOOTTMP    EQU   $3C
C600:          004F   BOOTDEV    EQU   $4F
C600:A2 20                       LDX   #$20
C602:A0 00                       LDY   #$00
C604:64 03                       STZ   $03
C606:64 3C                       STZ   $3C
C608:A9 60                       LDA   #$60
C60A:AA                          TAX
C60B:86 2B            DRV2ENT    STX   SLOTZ
C60D:85 4F                       STA   BOOTDEV
C60F:5A                          PHY                      ;Y=1 IF DRIVE 2 BOOT, ELSE Y=0
C610:BD 8E C0                    LDA   $C08E,X
C613:BD 8C C0                    LDA   $C08C,X
C616:7A                          PLY
C617:B9 EA C0                    LDA   $C0EA,Y            ;SELECT DRIVE 1 OR 2
C61A:BD 89 C0                    LDA   $C089,X
C61D:A0 50                       LDY   #$50
C61F:BD 80 C0         SEEKZERO   LDA   $C080,X
C622:98                          TYA
C623:29 03                       AND   #$03
C625:0A                          ASL   A
C626:05 2B                       ORA   SLOTZ
C628:AA                          TAX
C629:BD 81 C0                    LDA   $C081,X
C62C:A9 56                       LDA   #$56
C62E:20 A8 FC                    JSR   WAIT
C631:88                          DEY
C632:10 EB     C61F              BPL   SEEKZERO
C634:85 26                       STA   $26
C636:85 3D                       STA   $3D
C638:85 41                       STA   $41
C63A:20 09 C7                    JSR   MAKTBL
C63D:64 03            EXTENT1    STZ   $03
C63F:18               RDADR      CLC
C640:08                          PHP
C641:28               RETRY1     PLP
C642:A6 2B            RDDHDR     LDX   SLOTZ              ;RESTORE X TO $60
C644:C6 03                       DEC   $03                ;UPDATE RETRY COUNT
C646:D0 0E     C656              BNE   RDHD0              ;BRANCH IF NOT OUT OF RETRIES
C648:BD 88 C0         FUGIT      LDA   $C088,X            ;SHUT OFF DISK AND QUIT!
C64B:BD CF C6         FUG1       LDA   MSG-$60,X          ;(X STARTS AT $60)
C64E:10 FE     C64E   HANGING    BPL   HANGING            ;HANG, HANG, HANG!
C650:9D 7B 07                    STA   BOOTSCRN-$60,X
C653:E8                          INX
C654:80 F5     C64B              BRA   FUG1
C656:08               RDHD0      PHP
C657:88               RETRY      DEY
C658:D0 04     C65E              BNE   RDHD1
C65A:F0 E5     C641              BEQ   RETRY1
C65C:80 DF     C63D   EXTENT     BRA   EXTENT1
C65E:                 * * * * * * * * * * * * * * * * * * * * *
C65E:                 * The following code is sacred in it's  *
C65E:                 * present form.  To change it would     *
C65E:                 * cause volcanos to errupt, the ground  *
```

```
C65E:                 * to shake, and ProDOS not to boot!     *
C65E:                 * * * * * * * * * * * * * * * * * * * * *
C65E:BD 8C C0         RDHD1    LDA   $C08C,X
C661:10 FB    C65E             BPL   RDHD1
C663:49 D5            ISMRK1   EOR   #$D5
C665:D0 F0    C657             BNE   RETRY
C667:BD 8C C0         RDHD2    LDA   $C08C,X
C66A:10 FB    C667             BPL   RDHD2
C66C:C9 AA                     CMP   #$AA
C66E:D0 F3    C663             BNE   ISMRK1
C670:EA                        NOP
C671:BD 8C C0         RDHD3    LDA   $C08C,X
C674:10 FB    C671             BPL   RDHD3
C676:C9 96                     CMP   #$96
C678:F0 09    C683             BEQ   RDSECT
C67A:28                        PLP
C67B:90 C2    C63F             BCC   RDADR
C67D:49 AD                     EOR   #$AD
C67F:F0 25    C6A6             BEQ   RDATA
C681:D0 BC    C63F             BNE   RDADR
C683:A0 03            RDSECT   LDY   #$03
C685:85 40            RDSEC1   STA   $40
C687:BD 8C C0         RDSEC2   LDA   $C08C,X
C68A:10 FB    C687             BPL   RDSEC2
C68C:2A                        ROL   A
C68D:85 3C                     STA   BOOTTMP
C68F:BD 8C C0         RDSEC3   LDA   $C08C,X
C692:10 FB    C68F             BPL   RDSEC3
C694:25 3C                     AND   BOOTTMP
C696:88                        DEY
C697:D0 EC    C685             BNE   RDSEC1
C699:28                        PLP
C69A:C5 3D                     CMP   $3D
C69C:D0 A1    C63F             BNE   RDADR
C69E:A5 40                     LDA   $40
C6A0:C5 41                     CMP   $41
C6A2:D0 9B    C63F    BADRD1   BNE   RDADR
C6A4:B0 9C    C642             BCS   RDDHDR
C6A6:A0 56            RDATA    LDY   #$56
C6A8:84 3C            RDAT0    STY   BOOTTMP
C6AA:BC 8C C0         RDAT1    LDY   $C08C,X
C6AD:10 FB    C6AA             BPL   RDAT1
C6AF:59 D6 02                  EOR   DNIBL-$80,Y
C6B2:A4 3C                     LDY   BOOTTMP
C6B4:88                        DEY
C6B5:99 00 03                  STA   NBUF1,Y
C6B8:D0 EE    C6A8             BNE   RDAT0
C6BA:84 3C            RDAT2    STY   BOOTTMP
C6BC:BC 8C C0         RDAT3    LDY   $C08C,X
C6BF:10 FB    C6BC             BPL   RDAT3
C6C1:59 D6 02                  EOR   DNIBL-$80,Y
C6C4:A4 3C                     LDY   BOOTTMP
C6C6:91 26                     STA   ($26),Y
C6C8:C8                        INY
C6C9:D0 EF    C6BA             BNE   RDAT2
C6CB:BC 8C C0         RDAT4    LDY   $C08C,X
C6CE:10 FB    C6CB             BPL   RDAT4
C6D0:59 D6 02                  EOR   DNIBL-$80,Y
```

```
C6D3:D0 CD    C6A2  BADREAD   BNE   BADRD1
C6D5:A0 00                    LDY   #$00
C6D7:A2 56          DENIBL    LDX   #$56
C6D9:CA             DENIB1    DEX
C6DA:30 FB    C6D7            BMI   DENIBL
C6DC:B1 26                    LDA   ($26),Y
C6DE:5E 00 03                 LSR   NBUF1,X
C6E1:2A                       ROL   A
C6E2:5E 00 03                 LSR   NBUF1,X
C6E5:2A                       ROL   A
C6E6:91 26                    STA   ($26),Y
C6E8:C8                       INY
C6E9:D0 EE    C6D9            BNE   DENIB1
C6EB:               * * * * * * * * * * * * * * * * *
C6EB:               * Code beyond this point is not *
C6EB:               * sacred... It may be perverted *
C6EB:               * in any manner by any pervert. *
C6EB:               * * * * * * * * * * * * * * * * *
C6EB:E6 27                    INC   $27
C6ED:E6 3D                    INC   $3D
C6EF:A5 3D                    LDA   $3D
C6F1:CD 00 08                 CMP   $0800
C6F4:A6 4F                    LDX   BOOTDEV
C6F6:90 DB    C6D3            BCC   BADREAD
C6F8:4C 01 08                 JMP   $0801
C6FB:4C 0B C6       DODRV2    JMP   DRV2ENT
C6FE:         0002            DS    $C700-*,0
C700:FF                       DFB   $FF               ;MAKE IT LOOK LIKE NOTHING IN SLOT
C701:A9 E0          DRV2BOOT  LDA   #$E0              ;FOR DEVICE #2
C703:A0 01                    LDY   #1                ;TO SELECT DRIVE 2
C705:A2 60                    LDX   #$60
C707:80 F2    C6FB            BRA   DODRV2
C709:A2 03          MAKTBL    LDX   #$03
C70B:A0 00                    LDY   #0
C70D:86 3C          TBLLOOP   STX   BOOTTMP
C70F:8A                       TXA
C710:0A                       ASL   A
C711:24 3C                    BIT   BOOTTMP
C713:F0 10    C725            BEQ   NOPATRN
C715:05 3C                    ORA   BOOTTMP
C717:49 FF                    EOR   #$FF
C719:29 7E                    AND   #$7E
C71B:B0 08    C725  TBLLOOP2  BCS   NOPATRN
C71D:4A                       LSR   A
C71E:D0 FB    C71B            BNE   TBLLOOP2
C720:98                       TYA
C721:9D 56 03                 STA   DNIBL,X
C724:C8                       INY
C725:E8             NOPATRN   INX
C726:10 E5    C70D            BPL   TBLLOOP
C728:A9 08                    LDA   #$08
C72A:85 27                    STA   $27
C72C:A0 7F                    LDY   #$7F
C72E:60                       RTS
C72F:         C72F  MSG       EQU   *
C72F:                         MSB   ON
C72F:C3 E8 E5 E3              ASC   'Check         Disk Drive.'
C740:               *
```

```
C740:                * The following code is Teri's memory and
C740:                * soft switch exercise program.  The only
C740:                * purpose is exercise, not diagnostic
C740:                * functions.  This code is activated on
C740:                * a system without a keyboard, or when
C740:                * both open and closed apple keys are
C740:                * pressed during the reset sequence.
C740:                *
C740:08 50 52          TBL1      DFB    $08,$50,$52   ;These are low order
C743:00 02 04                    DFB    $00,$02,$04   ; addresses of $C0XX
C746:8B 8B E8                    DFB    $8B,$8B,$E8   ; that must be re-selected
C749:09 50 52          TBL2      DFB    $09,$50,$52   ; after each page write
C74C:00 03 05                    DFB    $00,$03,$05   ; (especially $C000!)
C74F:83 83 E8                    DFB    $83,$83,$E8
C752:                *
C752:64 00             XLOOP1    STZ    $00           ;Reset low address to 2
C754:E6 00                       INC    $00           ;Hi addr assumed to = 0
C756:E6 00                       INC    $00
C758:92 00             XPAGE     STA    ($00)         ;Write entire page with
C75A:9D 00 C0                    STA    $C000,X       ; shifted data... BUT
C75D:6A                          ROR    A             ; restore Z-page after
C75E:E6 00                       INC    $00           ; write in case $C008-9
C760:D0 F6    C758               BNE    XPAGE         ; is current pointer
C762:18                          CLC                   ;Indicates regular pass
C763:98                XMODE     TYA                   ;Get settings, each bit
C764:A0 08                       LDY    #$08          ;Specifies main/alt set
C766:BE 40 C7          XRSET     LDX    TBL1,Y        ;Assume Main $C000 setting
C769:90 03    C76E               BCC    XRST1         ;Branch if Main setting
C76B:BE 49 C7                    LDX    TBL2,Y        ;Else get Alternate index
C76E:9D 00 C0          XRST1     STA    $C000,X
C771:2A                          ROL    A             ;Accumulator makes full
C772:88                          DEY                  ; circle
C773:10 F1    C766               BPL    XRSET
C775:A8                          TAY                  ;Preserve settings in Y
C776:B0 DA    C752               BCS    XLOOP1        ;Branch if new setting
C778:E6 01                       INC    $01
C77A:D0 DC    C758               BNE    XPAGE         ;Loop til all pages writen
C77C:38                BANGER    SEC                  ;Indicate new settings,
C77D:C8                          INY                  ; reset mem pointer after
C77E:80 E3    C763               BRA    XMODE         ; after new settings
C780:         0000               DS     $C780-*
C780:                            INCLUDE MBASIC       ;Mouse BASIC routines @$C780
```

```
C780:                 ******************************************
C780:                 *
C780:                 * XMBASIC - Basic call to the mouse
C780:                 *
C780:                 ******************************************
C780:         C780    xmbasic    equ    *
C780:5A                          phy
C781:B0 1C    C79F               bcs    basicin        ;Input?
C783:A0 C4                       ldy    #<mbasic       ;Input from $C400?
C785:C4 39                       cpy    kswh
C787:D0 04    C78D               bne    xmbout
C789:A4 38                       ldy    kswl
C78B:F0 12    C79F               beq    basicin
C78D:DA               xmbout     phx                    ;Save X too
C78E:48                          pha
C78F:29 7F                       and    #$7F           ;We don't care about high bit
C791:C9 02                       cmp    #2
C793:B0 06    C79B               bge    mbbad          ;Only 0,1 valid
C795:20 3D C4                    jsr    xsetmou
C798:20 6D C4                    jsr    xmhome
C79B:68               mbbad      pla
C79C:FA                          plx
C79D:7A                          ply
C79E:60                          rts


C79F:                 ******************************************
C79F:                 *
C79F:                 * BASICIN - Input from basic
C79F:                 *
C79F:                 * Creates  +XXXXX,+YYYYY,+SS
C79F:                 *  XXXXX = X position
C79F:                 *  YYYYY = Y position
C79F:                 *  SS = Status
C79F:                 *       - = Key pressed
C79F:                 *       1 = Button pressed
C79F:                 *       2 = Button just pressed
C79F:                 *       3 = Button just released
C79F:                 *       4 = Button not pressed
C79F:                 *
C79F:                 ******************************************
C79F:         C79F    basicin    equ    *
C79F:91 28                       sta    (basl),y       ;Fix flashing char
C7A1:A9 05                       lda    #>inent        ;Fix input entry
C7A3:85 38                       sta    kswl
C7A5:AD 00 C0                    lda    kbd            ;test the keyboard
C7A8:0A                          asl    A
C7A9:08                          php                    ;Save kbd and int stat for later
C7AA:78                          sei                    ;No interrupts while getting position
C7AB:20 95 C4                    jsr    xmread
C7AE:A0 05                       ldy    #5             ;Move X position into the buffer
C7B0:AE 7C 05                    ldx    mouxh
C7B3:AD 7C 04                    lda    mouxl
C7B6:20 9B C5                    jsr    hextodec       ;Convert it
C7B9:A0 0C                       ldy    #12
C7BB:AE FC 05                    ldx    mouyh
C7BE:AD FC 04                    lda    mouyl
```

```
C7C1:20 9B C5                  jsr    hextodec
C7C4:AD 7C 07                  lda    moustat
C7C7:2A                        rol    A
C7C8:2A                        rol    A
C7C9:2A                        rol    A
C7CA:29 03                     and    #3
C7CC:49 03                     eor    #3
C7CE:1A                        inc    A
C7CF:28                        plp                     ;Restore int & kbd status
C7D0:A0 10                     ldy    #16
C7D2:20 AC C5                  jsr    hexdec2        ;X=0 from last div10
C7D5:7A                        ply
C7D6:A2 11                     ldx    #17            ;X = EOL
C7D8:A9 8D                     lda    #$8D           ;Carriage return
C7DA:9D 00 02       putinbuf   sta    inbuf,x
C7DD:60                        rts


C7DE:               *****************************************
C7DE:               *
C7DE:               * PADDLE patch
C7DE:               *
C7DE:               *****************************************
C7DE:         C7DE  mpaddle    equ    *
C7DE:AD FC 07                  lda    moumode        ;Is the mouse active?
C7E1:C9 01                     cmp    #01            ;Only transparent mode
C7E3:F0 06    C7EB             beq    pdon
C7E5:AD 70 C0                  lda    vblclr         ;Fire the strobe
C7E8:4C 21 FB                  jmp    $FB21
C7EB:         C7EB  pdon       equ    *
C7EB:E0 01                     cpx    #1             ;C=1 if X=1
C7ED:6A                        ror    A              ;A=80 or 0
C7EE:A8                        tay
C7EF:B9 7C 05                  lda    mouxh,y        ;Get high byte
C7F2:F0 02    C7F6             beq    pdok
C7F4:A9 FF                     lda    #$FF
C7F6:19 7C 04       pdok       ora    mouxl,y
C7F9:A8                        tay
C7FA:60                        rts
C7FB:5D E8 C5       zznml      eor    qtbl,x
C7FE:80 DA    C7DA             bra    putinbuf
C800:         0000             ds     $C800-*,0
C800:                          INCLUDE IRQBUF        ;Interrupt stuff @$C800
```

```
C800:                   *
C800:                   * this is the main (only) IRQ handling routines
C800:                   *
C800:4C E4 C1                      jmp    plinit        ;Pascal 1.0 Initialization
C803:48                 NEWIRQ     PHA                  ;SAVE ACC ON STACK, NOT $45
C804:68                            PLA                  ;LEGAL BECAUSE IF IRQ, IRQ DISABLED.
C805:68                            PLA                  ;GET STATUS REGISTER
C806:48                 IRQ1       PHA
C807:D8                            CLD                  ;CLEAR DEC MODE, ELSE THINGS GET SCREWED.
C808:29 10                         AND    #$10          ;SET CARRY TO INDICATE BRK
C80A:69 F0                         ADC    #$F0
C80C:8A                            TXA                  ;SAVE X IN A WHILE
C80D:BA                            TSX                  ; FUTZING WITH THE STACK
C80E:CA                            DEX                  ; RECOVER A-REG AT TOP...
C80F:9A                            TXS
C810:48                            PHA                  ;SAVE X ON STACK (ON TOP OF A)
C811:5A                            PHY                  ; AND Y ALSO
C812:AE 66 C0                      LDX    MOUX1         ;Get mouse info
C815:AC 67 C0                      LDY    MOUY1
C818:AD 18 C0                      LDA    RD80COL       ;TEST FOR 80-STORE WITH
C81B:2D 1C C0                      AND    RDPAGE2       ; PAGE 2 TEXT.
C81E:29 80                         AND    #$80          ; MAKE IT ZERO OR $80
C820:F0 05    C827                 BEQ    IRQ2
C822:8D 54 C0                      STA    TXTPAGE1
C825:A9 40                         LDA    #$40          ;SET PAGE 2 RESET BIT.
C827:2C 13 C0           IRQ2       BIT    RDRAMRD
C82A:10 05    C831                 BPL    IRQ3          ;BRANCH IF MAIN RAM READ
C82C:8D 02 C0                      STA    RDMAINRAM     ;ELSE, SWITCH IT IN
C82F:09 20                         ORA    #$20          ; AND RECORD THE EVENT!
C831:2C 14 C0           IRQ3       BIT    RDRAMWRT      ;DO THE SAME FOR RAM WRITE.
C834:10 05    C83B                 BPL    IRQ4
C836:8D 04 C0                      STA    WRMAINRAM
C839:09 10                         ORA    #$10
C83B:B0 13    C850      IRQ4       BCS    IRQ5          ;BRANCH IF BREAK, NOT INTERRUPT
C83D:48                            PHA                  ;SAVE MACHINE STATES SO FAR...
C83E:20 D5 C4                      JSR    MOUSEINT      ;GO TEST THE MOUSE
C841:90 3F    C882                 BCC    IRQDONE       ;BRANCH IF IT WAS THE MOUSE
C843:20 00 C9                      JSR    ACIAINT       ;GO TEST ACIA AND KEYBOARD INTERRUPTS
C846:90 3A    C882                 BCC    IRQDONE       ;BRANCH IF INTERRUPT SERVICED
C848:68                            PLA                  ;RESTORE STATES RECORDED SO FAR
C849:18                            CLC                  ;RESET BREAK/INTERRUPT INDICATOR
C84A:80 04    C850                 bra    passkipl      ;Skip around pascal 1.0 stuff
C84C:         0001                 ds     $C84D-*,$00
C84D:4C EE C1                      jmp    plread
C850:         C850      passkipl   equ    *
C850:2C 12 C0           IRQ5       BIT    RDLCRAM       ;DETERMINE IF LANGUAGE CARD ACTIVE
C853:10 0C    C861                 BPL    IRQ7
C855:09 0C                         ORA    #$C           ;SET TWO BITS SO RESTORED
C857:2C 11 C0                      BIT    RDLCBNK2      ; LANGUAGE CARD IS WRITE ENABLED
C85A:10 02    C85E                 BPL    IRQ6          ;BRANCH IF NOT PAGE 2 OF $D000
C85C:49 06                         EOR    #$6           ;ENABLE READ FOR PAGE 2 ON EXIT
C85E:8D 81 C0           IRQ6       STA    ROMIN
C861:2C 16 C0           IRQ7       BIT    RDALTZP       ;LAST...AND VERY IMPORTANT!
C864:10 0D    C873                 BPL    IRQ8          ; UNLESS IT IS NOT ENABLED
C866:BA                            TSX                  ;SAVE CURRENT STACK POINTER
C867:8E 01 01                      STX    $101          ;AT BOTTOM OF STACK
C86A:AE 00 01                      LDX    $100          ;GET MAIN STACK POINTER
C86D:9A                            TXS
```

```
C86E:8D 08 C0                    STA    SETSTDZP
C871:09 80                       ORA    #$80
C873:B0 2A     C89F  IRQ8        BCS    GOBREAK
C875:48                          PHA
C876:A9 C8                       LDA    #<IRQDONE
C878:48                          PHA
C879:A9 82                       LDA    #>IRQDONE        ;SAVE RETURN IRQ ADDR
C87B:48                          PHA
C87C:A9 04                       LDA    #4               ; SO WHEN INTERRUPT DOES RTI
C87E:48                          PHA                     ; IT RETURNS TO IRQDONE.
C87F:6C FE 03                    JMP    ($3FE)           ;PROCESS EXTERNAL INTERRUPT


C882:68              IRQDONE     PLA                     ;RECOVER MACHINE STATE
C883:10 07     C88C              BPL    IRQDNE1          ;BRANCH IF MAIN ZP WAS ACTIVE
C885:8D 09 C0                    STA    SETALTZP
C888:AE 01 01                    LDX    $101             ;RESTORE ALTERNATE STACK POINTER
C88B:9A                          TXS
C88C:0A              IRQDNE1     ASL    A
C88D:A0 05                       LDY    #$05
C88F:BE 89 C9        IRQDNE2     LDX    IRQTBLE,Y
C892:88                          DEY
C893:0A                          ASL    A
C894:90 03     C899              BCC    IRQDNE3          ;BRANCH IF SWITCH IS OK.
C896:9D 00 C0                    STA    $C000,X
C899:D0 F4     C88F  IRQDNE3     BNE    IRQDNE2          ;BRANCH IF MORE SWITCHES
C89B:7A                          PLY
C89C:FA                          PLX                     ;RESTORE ALL REGISTERS
C89D:68                          PLA
C89E:40                          RTI                     ;DO THE REAL RTI!
C89F:4C 47 FA        GOBREAK     JMP    NEWBRK           ;PASS THE BREAKER THROUGH



C8A2:                ******************************************
C8A2:                *
C8A2:                * MOVEIRQ - This routine transfers the roms interrupt vector into
C8A2:                *  both language cards
C8A2:                *
C8A2:                ******************************************
C8A2:          C8A2  moveirq     equ    *
C8A2:20 60 C3                    JSR    SETROM           ;Read ROM and Write to RAM
C8A5:AD 16 C0                    LDA    RDALTZP          ;Which language card?
C8A8:0A                          ASL    A                ;C=1 if alternate card
C8A9:A0 01                       LDY    #1               ;Move two bytes
C8AB:B9 FE FF        MIRQLP      LDA    IRQVECT,Y        ;Get byte from ROM
C8AE:8D 09 C0                    STA    SETALTZP         ;Set alternate card
C8B1:99 FE FF                    STA    IRQVECT,Y        ;Store it in the RAM card
C8B4:8D 08 C0                    STA    SETSTDZP         ;Set main card
C8B7:99 FE FF                    STA    IRQVECT,Y
C8BA:88                          DEY
C8BB:10 EE     C8AB              BPL    MIRQLP           ;Go do the second byte
C8BD:90 03     C8C2              BCC    MIRQSTD          ;Is the card set right?
C8BF:8D 09 C0                    STA    SETALTZP         ;No, it wasn't
C8C2:4C 54 C3        MIRQSTD     JMP    RESETLC          ;Clean up & go home
```

```
C8C5:               *   This is the serial input routine.  Carry
C8C5:               * flag set indicates that returned data is
C8C5:               * valid.
C8C5:               *
C8C5:               * GETBUF- Gets a byte from the buffer & updates pointers
C8C5:               *  On entry Y=0 for Serial buffer Y=$80 for Keyboard buffer
C8C5:EC FF 04       XRDSER   CPX   ACIABUF      ;is serial input buffered?
C8C8:D0 26    C8F0           BNE   XNOSBUF      ;(in english "NO SERIAL BUFFER")
C8CA:A0 00                   LDY   #0           ;Y=0 for serial buffer
C8CC:         C8CC  GETBUF   EQU   *
C8CC:B9 7F 06                LDA   TRSER,Y      ;Test for data in buffer
C8CF:D9 7F 05                CMP   TWSER,Y      ;If = then no data
C8D2:F0 24    C8F8           BEQ   GBEMPTY
C8D4:48                      PHA                ;Save current value
C8D5:1A                      INC   A            ;Update the pointer
C8D6:89 7F                   BIT   #$7F         ;Overflow
C8D8:D0 01    C8DB           BNE   GBNOOVR
C8DA:98                      TYA
C8DB:99 7F 06       GBNOOVR  STA   TRSER,Y      ;Store the updated pointer
C8DE:7A                      PLY                ;Get the old value of the pointer
C8DF:AD 13 C0                LDA   RDRAMRD      ;Are we in main ram
C8E2:0A                      ASL   A            ;C=1 for Aux ram
C8E3:8D 03 C0                STA   RDCARDRAM    ;Force Aux ram
C8E6:B9 00 08                LDA   THBUF,Y      ;Get byte from buffer
C8E9:B0 14    C8FF           BCS   XRDSNO       ;Branch if we were in aux bank
C8EB:8D 02 C0                STA   RDMAINRAM    ;Set back to main
C8EE:38                      SEC
C8EF:60                      RTS                ;Note C=1
C8F0:               *
C8F0:BC 85 C8       XNOSBUF  LDY   DEVNO,X      ;Get index to ACIA
C8F3:B9 F9 BF                LDA   SSTAT,Y      ;Test ACIA directly for data
C8F6:29 08                   AND   #$8
C8F8:18             GBEMPTY  CLC                ;indicate no data
C8F9:F0 04    C8FF           BEQ   XRDSNO       ;Branch if no data!
C8FB:B9 F8 BF                LDA   SDATA,Y      ;get serial input
C8FE:38             notacia  SEC                ;indicate valid data returned.
C8FF:         C8FF  acdone   equ   *
C8FF:60             XRDSNO   RTS
```

```
C900:                 *  This routine will determine if the source of
C900:                 * is either of the built in ACIAs.  If neither port
C900:                 * generated the interrupt, or the interrupt was due
C900:                 * to a transmit buffer empty, protocol converter, or
C900:                 * 'unbuffered' receiver full, the carry is set indi-
C900:                 * cating an externally serviced interrupt.
C900:                 *  If the interrupt source was keyboard, 'buffered'
C900:                 * serial input, or the DCD, the interrupt is serviced
C900:                 * and the carry is cleared indicating interrupt was
C900:                 * serviced. (DCD handshake replaces CTS.)
C900:                 *  Location "ACIABUF" specifies which (if either) re-
C900:                 * ceiver data is buffered.  For port 1 it must contain
C900:                 * $C1, for port 2 a $C2.  Any other values are cause
C900:                 * interrupts to pass to external (RAM based) routines.
C900:                 *  Location "TYPHED" specifies whether Keyboard in-
C900:                 * put should be buffered, ignored, or processed by
C900:                 * RAM based routines.  If bit 7=1 and bit 6=0, key-
C900:                 * board data is placed in the type-ahead buffer. If
C900:                 * bit 6 is set the interrupt is cleared, but must
C900:                 * be recognized and serviced by a RAM routine.  If
C900:                 * both bits = 0, the interrupt is serviced, but the
C900:                 * keyboard data is ignored.
C900:                 *  While using type-ahead, Open-Apple CTRL-X will
C900:                 * flush the buffer.  No other code is recognized.
C900:                 *  If the source was an ACIA that has the transmit
C900:                 * interrupt enabled, the original value of the ACIAs
C900:                 * status registers is preserved. Automatic serial input
C900:                 * buffering is not serviced from a port so configured.
C900:                 * Interrupts originating from the protocol converter or
C900:                 * keyboard (RAM serviced) do not inhibit serial buffering
C900:                 * and are passed thru.  The RAM service routine can rec-
C900:                 * ognize the interrupt source by a 1 state in bit 6 of
C900:                 * the ACIAs status register.  The RAM service routine must
C900:                 * cause the clearing of DSR (bit 6) AND make a second ac-
C900:                 * cess to the status register before returning.
C900:                 *
C900:                 *
C900:           C900  aciaint  equ  *
C900:A2 C2                     ldx  #<comslot      ;Test port 2 first
C902:20 08 C9                  jsr  aciatst        ;Check for interrupt
C905:90 F8      C8FF           bcc  acdone         ;Return if interrupt done
C907:CA                        dex                  ;Try port 1
C908:BC 85 C8         aciatst  ldy  devno,x        ;Get index for acia
C90B:A9 04                     lda  #$4            ;If xmit ints enabled pass to user
C90D:59 FA BF                  eor  scomd,y        ;Check if D<3>, D<2> = 01
C910:29 0C                     and  #$0C           ;
C912:F0 EA      C8FE           beq  notacia        ;User better take it!
C914:B9 F9 BF                  lda  sstat,y        ;Get status
C917:9D 38 04                  sta  astat,x        ;Save it away
C91A:10 E2      C8FE           bpl  notacia        ;No interrupt
C91C:E0 C2            aitst2   cpx  #<comslot      ;C=1 if com port
C91E:B0 02      C922           bcs  aiport2        ;Invert DSR if port1
C920:49 40                     eor  #$40
C922:3C 38 05         aiport2  bit  extint,x       ;Is DSR enabled?
C925:70 26      C94D           bvs  aipass         ;Yes, user wants it
C927:10 22      C94B           bpl  aieatit        ;No, eat it
C929:90 20      C94B           bcc  aieatit        ;Yes but I don't want it for port 1
C92B:89 40                     bit  #$40           ;Is DSR 1?
```

```
C92D:F0 1E   C94D            beq   aipass        ;If not, skip it
C92F:             * It's a keyboard interrupt
C92F:AD 00 C0                lda   kbd           ;Get the key
C932:A0 80                   ldy   #$80
C934:20 67 C9                jsr   putbuf        ;Put it in the buffer
C937:C9 98                   cmp   #$98          ;Is it a ^x?
C939:D0 08   C943            bne   ainoflsh
C93B:AD 62 C0                lda   butnl         ;And the closed apple?
C93E:10 03   C943            bpl   ainoflsh
C940:20 1B CB                jsr   flush         ;Flush the buffer
C943:AD 10 C0     ainoflsh   lda   kbdstrb       ;Clear the keyboard
C946:             * $A0 $B0 table needed by serial firmware
C946:        00C1 sltdmy     equ   <serslot
C946:        C885 devno      equ   *-sltdmy
C946:A0 B0                   ldy   #$B0          ;Restore y
C948:B9 F9 BF                lda   sstat,y       ;Read status to clear int
C94B:29 BF        aieatit    and   #$BF          ;Clear the DSR bit
C94D:0A           aipass     asl   A             ;Shift DSR into C
C94E:0A                      asl   A
C94F:29 20                   and   #$20          ;Is the receiver full?
C951:F0 35   C988            beq   aciadone      ;If not, we're done
C953:B9 FA BF                lda   scomd,y       ;Are receive interrupts enabled?
C956:49 01                   eor   #1            ;Check for D<1>,D<0> = 01
C958:29 03                   and   #3
C95A:D0 2C   C988            bne   aciadone      ;If not, were done
C95C:8A                      txa                  ;Is this acia buffered?
C95D:4D FF 04                eor   aciabuf
C960:D0 9C   C8FE            bne   notacia       ;The user better handle it!
C962:B9 F8 BF                lda   sdata,y       ;It's mine
C965:A0 00                   ldy   #0
C967:        C967 putbuf     equ   *
C967:DA                      phx
C968:48                      pha
C969:B9 7F 05                lda   twser,y       ;Get buffer pointer
C96C:AA                      tax                  ;Save it for later
C96D:1A                      inc   A             ;Bump it to next free byte
C96E:89 7F                   bit   #$7F          ;Overflow?
C970:D0 01   C973            bne   pbok
C972:98                      tya                  ;Wrap pointer
C973:08           pbok       php                  ;Save DSR status
C974:D9 7F 06                cmp   trser,y       ;Buffer full?
C977:F0 03   C97C            beq   pbfull
C979:99 7F 05                sta   twser,y       ;Save the new pointer
C97C:28           pbfull     plp
C97D:68                      pla                  ;Get the data
C97E:8D 05 C0                sta   wrcardram     ;It goes to aux ram
C981:9D 00 08                sta   thbuf,x
C984:8D 04 C0                sta   wrmainram
C987:FA                      plx
C988:60           aciadone   rts


C989:83 8B 8B     IRQTBLE    DFB   >LCBANK2,>LCBANK1,>LCBANK1
C98C:05 03 55                DFB   >WRCARDRAM,>RDCARDRAM,TXTPAGE2
```

```
C98F:                *   The following two routines are for reading key-
C98F:                * board and serial input from buffers or directly.
C98F:                *   Type-ahead buffering only occurs for non auto-
C98F:                * repeat keypresses.  When a key is pressed for
C98F:                * auto-repeat the buffer is first emptied, then the
C98F:                * repeated characters are returned.
C98F:                *   The minus flag is used to indicate if a keystroke
C98F:                * is being returned.
C98F:                *

C98F:20 AD C9        XRDKBD     JSR    XBITKBD         ;is keyboard input ready?
C992:10 14    C9A8              BPL    XNOKEY          ;Branch if not.
C994:90 0A    C9A0              BCC    XRKBD1          ;Branch if direct KBD input.
C996:5A                         PHY                    ;Save Y
C997:A0 80                      LDY    #$80            ;Y=$80 for keyboard buffer
C999:20 CC C8                   JSR    GETBUF          ;Get data from buffer
C99C:7A                         PLY
C99D:09 00                      ORA    #0              ;Set minus flag
C99F:60                         RTS

C9A0:AD 00 C0        XRKBD1     LDA    KBD             ;test keyboard directly
C9A3:10 EA    C98F              BPL    XRDKBD          ;loop if buffered since test.
C9A5:8D 10 C0                   STA    KBDSTRB         ;Clear keyboard strobe.
C9A8:60              XNOKEY     RTS                    ;Minus flag indicates valid character

C9A9:         0001              ds     $C9AA-*,$00
C9AA:4C F6 C1                   jmp    plwrite         ;Pascal 1.0 entry point

C9AD:2C FA 05        XBITKBD    BIT    TYPHED          ;This routine replaces "BIT KBD" instrucs
C9B0:10 10    C9C2              BPL    XBKB2           ; so as to function with type-ahead.
C9B2:38                         SEC                    ;anticipate data in buffer is ready
C9B3:08                         PHP                    ;save carry and minus flags
C9B4:48                         PHA                    ;preserve accumulator
C9B5:AD FF 06                   LDA    TRKEY
C9B8:CD FF 05                   CMP    TWKEY           ;is there data to be read?
C9BB:F0 03    C9C0              BEQ    XBKB1           ;branch if type-ahead buffer empty
C9BD:68                         PLA
C9BE:28                         PLP
C9BF:60                         RTS                    ;Carry and minus flag already set.
C9C0:                *
C9C0:68              XBKB1      PLA
C9C1:28                         PLP                    ;restore ACC and Status
C9C2:2C 00 C0        XBKB2      BIT    KBD             ;test KBD Directly
C9C5:18                         CLC                    ;indicate direct test
C9C6:60                         RTS
C9C7:                           INCLUDE COMMAND        ;Serial firmware command processor
```

```
C9C7:                           MSB   OFF
C9C7:             C9C7 cmdtable equ   *
C9C7:66                         dfb   >cmdi-1
C9C8:66                         dfb   >cmdk-1
C9C9:66                         dfb   >cmdl-1
C9CA:5C                         dfb   >cmdn-1
C9CB:5C                         dfb   >cmdcr-1
C9CC:7C                         dfb   >cmdb-1
C9CD:78                         dfb   >cmdd-1
C9CE:77                         dfb   >cmdp-1
C9CF:C3                         dfb   >cmdq-1
C9D0:B4                         dfb   >cmdr-1
C9D1:98                         dfb   >cmds-1
C9D2:C5                         dfb   >cmdt-1
C9D3:54                         dfb   >cmdz-1
C9D4:7F BF BF 7F       mask1    dfb   $7F,$BF,$BF,$7F,$FF
C9D9:80 00 40 00       mask2    dfb   $80,$00,$40,$00,$00
C9DE:             C9DE cmdlist  equ   *
C9DE:49 4B 4C 4E                asc   "IKLN"
C9E2:0D                         dfb   $0D            ;Carriage return
C9E3:42 44 50 51                asc   "BDPQRSTZ"
C9EB:48                command  pha                   ;Check for command to firmware
C9EC:3C B8 03                   bit   sermode,x      ;Already in command?
C9EF:30 1B        CA0C          bmi   incmd          ;If so,go do it
C9F1:BC 38 06                   ldy   eschar,x       ;If eschar = 0 ignore commands
C9F4:F0 13        CA09          beq   nocmd
C9F6:5D 38 06                   eor   eschar,x       ;Is it the command char?
C9F9:0A                         asl   A              ;Ignore high bit
C9FA:D0 0D        CA09          bne   nocmd
C9FC:AC FB 07                   ldy   cursor         ;Save the cursor
C9FF:8C 79 06                   sty   oldcur
CA02:A0 BF                      ldy   #cmdcur        ;Set command cursor
CA04:8C FB 07                   sty   cursor
CA07:80 2D        CA36          bra   cominit
CA09:38                nocmd    sec                   ;Mark char not handled
CA0A:68                         pla                   ;Restore char
CA0B:60                         rts
CA0C:             CA0C incmd    equ   *              ;Command mode
CA0C:BC 85 C8                   ldy   devno,x        ;Get index for ACIA
CA0F:29 5F                      and   #$5F           ;Ignore hi bit: just upshift lowercase
CA11:DA                         phx                   ;Save slot
CA12:A2 0C                      ldx   #12            ;Check 13 commands
CA14:DD DE C9          cmdloop  cmp   cmdlist,x
CA17:F0 34        CA4D          beq   cmfound        ;Right char?
CA19:CA                         dex
CA1A:10 F8        CA14          bpl   cmdloop
CA1C:FA                         plx                   ;We didn't find it
CA1D:68                         pla
CA1E:48                         pha
CA1F:29 7F                      and   #$7F           ;if char is cntl char
CA21:C9 20                      cmp   #$20           ;it can be the new comd char
CA23:B0 03        CA28          bcs   ckdig          ;branch if not cntl character
CA25:9D 38 06          cmdz2    sta   eschar,x       ;Save comd char, drop thru ckdig to cdone
CA28:49 30             ckdig    eor   #$30           ;Is it a number?
CA2A:C9 0A                      cmp   #$0A
CA2C:B0 0E        CA3C          bcs   cdone          ;If so, branch
CA2E:A0 0A                      ldy   #10            ;A = A + 10 * current number
CA30:6D 7F 04          digloop  adc   number         ;C=0 on first entry
```

```
CA33:88                          dey
CA34:D0 FA   CA30                bne   digloop
CA36:8D 7F 04         cominit    sta   number
CA39:38                          sec                   ;Mark in command mode
CA3A:80 07   CA43                bra   cmset
CA3C:18               cdone      clc                   ;Out of command mode
CA3D:AD 79 06                    lda   oldcur         ;Restore the cursor
CA40:8D FB 07                    sta   cursor
CA43:08               cmset      php
CA44:1E B8 03                    asl   sermode,x      ;set command mode according to carry
CA47:28                          plp
CA48:7E B8 03                    ror   sermode,x       ;leaves carry clear
CA4B:68                          pla                    ;character handled
CA4C:60                          rts                  ;because carry clear...

CA4D:A9 CA            cmfound    lda   #<cmdcr
CA4F:48                          pha                  ;do JMP via RTS
CA50:BD C7 C9                    lda   cmdtable,x
CA53:48                          pha
CA54:60                          rts                  ;Go to it

CA55:FA               cmdz       plx                   ;Zero escape character
CA56:9E B8 04                    stz   pwdth,x        ;And the width
CA59:A9 00                       lda   #0
CA5B:80 C8   CA25                bra   cmdz2

CA5D:        CA5D     cmdcr      equ   *
CA5D:        CA5D     cmdn       equ   *
CA5D:7A                          ply
CA5E:AD 7F 04                    lda   number         ;Get number inputted
CA61:F0 05   CA68                beq   cmdi2          ;Don't change printer width if 0
CA63:99 B8 04                    sta   pwdth,y        ;Update printer width
CA66:F0                          dfb   $F0            ;BEQ opcode to skip next byte
CA67:        CA67     cmdi       equ   *
CA67:        CA67     cmdk       equ   *
CA67:        CA67     cmdl       equ   *
CA67:7A                          ply
CA68:B9 B8 06         cmdi2      lda   flags,y
CA6B:3D D4 C9                    and   mask1,x        ;Mask off bit we'll change
CA6E:1D D9 C9                    ora   mask2,x        ;Change it
CA71:99 B8 06                    sta   flags,y        ;Back it goes
CA74:98                          tya                   ;Put slot back in x
CA75:AA                          tax
CA76:80 C4   CA3C     cdone2     bra   cdone          ;Good bye

CA78:88               cmdp       dey                   ;Make y point to command reg
CA79:A9 1F            cmdd       lda   #$1F           ;Mask off high three bits
CA7B:38                          sec                   ;C=1 means high 3 bits
CA7C:90                          dfb   $90            ;BCC opcode to skip next byte
CA7D:A9 F0            cmdb       lda   #$F0           ;Mask off lower 4 bits F0 = BNE
CA7F:18                          clc                   ;F0 will skip this if cmdp or cmdd
CA80:39 FB BF                    and   scntl,y        ;Mask off bits being changed
```

```
CA83:8D F8 06                     sta  temp        ;Save it
CA86:FA                           plx
CA87:AD 7F 04                     lda  number      ;Get inputed number
CA8A:29 0F                        and  #$0F        ;Only lower nibble valid
CA8C:90 05    CA93                bcc  noshift     ;If C=1 shift to upper 3 bits
CA8E:0A                           asl  A
CA8F:0A                           asl  A
CA90:0A                           asl  A
CA91:0A                           asl  A
CA92:0A                           asl  A
CA93:0D F8 06         noshift     ora  temp        ;Get the rest of the bits
CA96:C8                           iny              ;Put them in the ACIA
CA97:80 17    CAB0                bra  cmdp2       ;increment puts em away where they go.


CA99:B9 FA BF         cmds        lda  scomd,y     ;Transmit a break
CA9C:48                           pha               ;Save current ACIA state
CA9D:09 0C                        ora  #$0C        ;Do the braek
CA9F:99 FA BF                     sta  scomd,y
CAA2:A9 E9                        lda  #233        ;For 233 ms
CAA4:A2 53            mswait      ldx  #83         ;Wait 1 ms
CAA6:48               msloop      pha              ;((12*82)+11)+2+3=1000us
CAA7:68                           pla
CAA8:CA                           dex
CAA9:D0 FB    CAA6                bne  msloop
CAAB:3A                           dec  a
CAAC:D0 F6    CAA4                bne  mswait
CAAE:68                           pla
CAAF:FA                           plx
CAB0:99 FA BF         cmdp2       sta  scomd,y
CAB3:80 C1    CA76                bra  cdone2


CAB5:99 F9 BF         cmdr        sta  sstat,y     ;Reset the ACIA
CAB8:AD 7B 06                     lda  vfactv      ;Check if video firmware active
CABB:0A                           asl  A           ;Save it in C
CABC:20 23 CE                     jsr  sethooks    ;assume video firmware active
CABF:90 03    CAC4                bcc  cmdq        ;branch if good guesser...
CAC1:20 4D CE                     jsr  zzquit      ;Reset the hooks
CAC4:18               cmdq        clc               ;Quit terminal mode
CAC5:B0                           dfb  $B0         ;BCS to skip next byte
CAC6:38               cmdt        sec               ;Into terminal mode
CAC7:FA                           plx               ;Recover X
CAC8:20 CD CA                     jsr  setterm
CACB:80 A9    CA76                bra  cdone2
CACD:BD B8 03         setterm     lda  sermode,x   ;Get terminal mode status
CAD0:89 40                        bit  #$40        ;Z=1 if not in terminal mode
CAD2:90 12    CAE6                bcc  stclr       ;Branch if clearing terminal mode
CAD4:D0 20    CAF6                bne  stwasok     ;Was already set
CAD6:E4 39                        cpx  kswh        ;Are we in the input hooks
CAD8:D0 47    CB21                bne  strts       ;Leaves C=1 if =
CADA:09 40                        ora  #$40        ;Set term mode bit
CADC:AC 79 06                     ldy  oldcur      ;Save what was in oldcur
CADF:8C 7A 06                     sty  oldcur2
CAE2:A0 DF                        ldy  #termcur    ;Get new cursor value
CAE4:80 07    CAED                bra  stset
```

```
CAE6:F0 0E    CAF6  stclr     beq   stwasok        ;Branch if already clear
CAE8:29 BF                    and   #$BF           ;Clear the bit
CAEA:AC 7A 06                 ldy   oldcur2        ;Restore the cursor
CAED:9D B8 03       stset     sta   sermode,x
CAF0:8C 79 06                 sty   oldcur         ;Save cursor to be restored after command
CAF3:8C FB 07                 sty   cursor
CAF6:BC 85 C8       stwasok   ldy   devno,x
CAF9:58                       cli                  ;want to leave with interrupts active
CAFA:08                       php
CAFB:78                       sei                  ;but off while we twittle bits
CAFC:B9 FA BF                 lda   scomd,y
CAFF:09 02                    ora   #$2            ;disable receiver interrupts if
CB01:90 02    CB05            bcc   cmdt2          ; not in terminal mode
CB03:29 FD                    and   #$FD           ;enable when in terminal mode
CB05:99 FA BF       cmdt2     sta   scomd,y
CB08:A9 00                    lda   #0
CB0A:6A                       ror   a              ;set kbd interrupts according to t-mode
CB0B:8D FA 05                 sta   typhed
CB0E:10 07    CB17            bpl   cmdt3          ;branch if leaving terminal mode
CB10:9C 7F 05                 stz   twser          ; and ser buf...
CB13:9C 7F 06                 stz   trser
CB16:8A                       txa                  ;use x to enable serial buffering
CB17:8D FF 04       cmdt3     sta   aciabuf
CB1A:28                       plp                  ;restore carry, enable interrupts.
CB1B:8E FF 05       flush     stx   twkey          ;Flush the type ahead buffer
CB1E:8E FF 06                 stx   trkey
CB21:60             strts     rts
CB22:         0002            ds    $CB24-*,$00
CB24:E8             zznm2     inx
CB25:4C FB C7                 jmp   zznml
CB28:9E 0B 40 50    comtbl    dfb   $9E,$0B,$40,$50,$16,$0B,$01,$00
CB30:                         INCLUDE SCROLLING    ;More Video stuff @$CB30
```

```
CB30:                 *
CB30:                 * SCROLLIT scrolls the screen either up or down, depending
CB30:                 * on the value of X.  It scrolls within windows with even
CB30:                 * or odd edges for both 40 and 80 columns.  It can scroll
CB30:                 * windows down to 1 characters wide.
CB30:                 *
CB30:DA               SCROLLDN  PHX                ;save X
CB31:A2 00                      LDX   #0           ;direction = down
CB33:80 03    CB38              BRA   SCROLLIT     ;do scroll
CB35:                 *
CB35:DA               SCROLLUP  PHX                ;save X
CB36:A2 01                      LDX   #1           ;direction = up
CB38:A4 21            SCROLLIT  LDY   WNDWDTH      ;get width of screen window
CB3A:2C 1F C0                   BIT   RD80VID      ;in 40 or 80 columns?
CB3D:10 18    CB57              BPL   GETST        ;=>40, determine starting line
CB3F:8D 01 C0                   STA   SET80COL     ;make sure this is enabled
CB42:98                         TYA                ;get WNDWDTH for test
CB43:4A                         LSR   A            ;divide by 2 for 80 column index
CB44:A8                         TAY                ;and save
CB45:A5 20                      LDA   WNDLFT       ;test oddity of right edge
CB47:4A                         LSR   A            ;by rotating low bit into carry
CB48:B8                         CLV                ;V=0 if left edge even
CB49:90 03    CB4E              BCC   CHKRT        ;=>check right edge
CB4B:2C C1 CB                   BIT   SEV1         ;V=1 if left edge odd
CB4E:2A               CHKRT     ROL   A            ;restore WNDLFT
CB4F:45 21                      EOR   WNDWDTH      ;get oddity of right edge
CB51:4A                         LSR   A            ;C=1 if right edge even
CB52:70 03    CB57              BVS   GETST        ;if odd left, don't DEY
CB54:B0 01    CB57              BCS   GETST        ;if even right, don't DEY
CB56:88                         DEY                ;if right edge odd, need one less
CB57:8C F8 05         GETST     STY   TEMPY        ;save window width
CB5A:AD 1F C0                   LDA   RD80VID      ;N=1 if 80 columns
CB5D:08                         PHP                ;save N,Z,V
CB5E:A5 22                      LDA   WNDTOP       ;assume scroll from top
CB60:E0 00                      CPX   #0           ;up or down?
CB62:D0 03    CB67              BNE   SETDBAS      ;=>up
CB64:A5 23                      LDA   WNDBTM       ;down, start scrolling at bottom
CB66:3A                         DEC   A            ;really need one less
CB67:                 *
CB67:8D 78 05         SETDBAS   STA   TEMPA        ;save current line
CB6A:20 24 FC                   JSR   VTABZ        ;calculate base with window width
CB6D:                 *
CB6D:A5 28            SCRLIN    LDA   BASL         ;current line is destination
CB6F:85 2A                      STA   BAS2L
CB71:A5 29                      LDA   BASH
CB73:85 2B                      STA   BAS2H
CB75:                 *
CB75:AD 78 05                   LDA   TEMPA        ;get current line
CB78:E0 00                      CPX   #0           ;going up?
CB7A:D0 07    CB83              BNE   SETUP2       =>up, inc current line
CB7C:C5 22                      CMP   WNDTOP       ;down. Reached top yet?
CB7E:F0 39    CBB9              BEQ   SCRL3        ;yes! clear top line, exit
CB80:3A                         DEC   A            ;no, go up a line
CB81:80 05    CB88              BRA   SETSRC       ;set source for scroll
CB83:1A               SETUP2    INC   A            ;up, inc current line
CB84:C5 23                      CMP   WNDBTM       ;at bottom yet?
CB86:B0 31    CBB9              BCS   SCRL3        ;yes! clear bottom line, exit
CB88:                 *
```

```
CB88:8D 78 05          SETSRC   STA   TEMPA          ;save new current line
CB8B:20 24 FC                   JSR   VTABZ          ;get base for new current line
CB8E:AC F8 05                   LDY   TEMPY          ;get width for scroll
CB91:28                         PLP                  ;get status for scroll
CB92:08                         PHP                  ;N=1 if 80 columns
CB93:10 1F    CBB4              BPL   SKPRT          ;=>only do 40 columns
CB95:AD 55 C0                   LDA   TXTPAGE2       ;scroll aux page first (even bytes)
CB98:98                         TYA                  ;test Y
CB99:F0 07    CBA2              BEQ   SCRLFT         ;if Y=0, only scroll one byte
CB9B:B1 28             SCRLEVEN LDA   (BASL),Y
CB9D:91 2A                      STA   (BAS2L),Y
CB9F:88                         DEY
CBA0:D0 F9    CB9B              BNE   SCRLEVEN       ;do all but last even byte
CBA2:70 04    CBA8     SCRLFT   BVS   SKPLFT         ;odd left edge, skip this byte
CBA4:B1 28                      LDA   (BASL),Y
CBA6:91 2A                      STA   (BAS2L),Y
CBA8:AD 54 C0          SKPLFT   LDA   TXTPAGE1       ;now do main page (odd bytes)
CBAB:AC F8 05                   LDY   TEMPY          ;restore width
CBAE:B0 04    CBB4              BCS   SKPRT          ;even right edge, skip this byte
CBB0:B1 28             SCRLODD  LDA   (BASL),Y
CBB2:91 2A                      STA   (BAS2L),Y
CBB4:88                SKPRT    DEY
CBB5:10 F9    CBB0              BPL   SCRLODD
CBB7:80 B4    CB6D              BRA   SCRLIN         ;scroll next line
CBB9:                  *
CBB9:20 A0 FC          SCRL3    JSR   CLRLIN         ;clear current line
CBBC:20 22 FC                   JSR   VTAB           ;restore original cursor line
CBBF:28                         PLP                  ;pull status off stack
CBC0:FA                         PLX                  ;restore X
CBC1:60                SEV1     RTS                  ;done!!!
```

```
CBC2:                  *
CBC2:                  * DOCLR is called by CLREOL.  It decides whether
CBC2:                  * to do a (quick) 40 or 80 column clear to end of line.
CBC2:                  *
CBC2:2C 1F C0          DOCLR    BIT   RD80VID      ;40 or 80 column clear?
CBC5:30 13    CBDA              BMI   CLR80        ;=>clear 80 columns
CBC7:91 28             CLR40    STA   (BASL),Y
CBC9:C8                         INY
CBCA:C4 21                      CPY   WNDWDTH
CBCC:90 F9    CBC7              BCC   CLR40
CBCE:60                         RTS
CBCF:                  *
CBCF:DA                CLRHALF  PHX                ;clear right half of screen
CBD0:A2 D8                      LDX   #$D8         ;for SCRN48
CBD2:A0 14                      LDY   #20
CBD4:A5 32                      LDA   INVFLG
CBD6:29 A0                      AND   #$A0
CBD8:80 17    CBF1              BRA   CLR2         ;=>jump into middle
CBDA:                  *
CBDA:DA                CLR80    PHX                ;preserve X
CBDB:48                         PHA                ;and blank
CBDC:98                         TYA                ;get count for CH
CBDD:48                         PHA                ;save for left edge check
CBDE:38                         SEC                ;count=WNDWDTH-Y-1
CBDF:E5 21                      SBC   WNDWDTH
CBE1:AA                         TAX                ;save CH counter
CBE2:98                         TYA                ;div CH by 2 for half pages
CBE3:4A                         LSR   A
CBE4:A8                         TAY
CBE5:68                         PLA                 ;restore original CH
CBE6:45 20                      EOR   WNDLFT       ;get starting page
CBE8:6A                         ROR   A
CBE9:B0 03    CBEE              BCS   CLR0
CBEB:10 01    CBEE              BPL   CLR0
CBED:C8                         INY                ;iff WNDLFT odd, starting byte odd
CBEE:68                CLR0     PLA                ;get blankity blank
CBEF:B0 0B    CBFC              BCS   CLR1         ;starting page is 1 (default)
CBF1:2C 55 C0          CLR2     BIT   TXTPAGE2     ;else do page 2
CBF4:91 28                      STA   (BASL),Y
CBF6:2C 54 C0                   BIT   TXTPAGE1     ;now do page 1
CBF9:E8                         INX
CBFA:F0 06    CC02              BEQ   CLR3         ;all done
CBFC:91 28             CLR1     STA   (BASL),Y
CBFE:C8                         INY                ;forward 2 columns
CBFF:E8                         INX                ;next CH
CC00:D0 EF    CBF1              BNE   CLR2         ;not done yet
CC02:FA                CLR3     PLX                ;restore X
CC03:60                         RTS                ;and exit
CC04:                  *
CC04:9C FA 05          CLRPORT  STZ   TYPHED       ;disable typeahead
CC07:9C F9 05                   STZ   EXTINT2      ;and external interrupts
CC0A:60                         RTS
```

```
CC0B:               *
CC0B:               * PASINVERT is used by Pascal to display the cursor. Pascal
CC0B:               * normally leaves the cursor on the screen at all times. It
CC0B:               * is fleetingly removed while a character is displayed, then
CC0B:               * promptly redisplayed.  CTL-F and CTL-E, respectively,
CC0B:               * disable and enable display of the cursor when printed using
CC0B:               * the Pascal 1.1 entry point (PWRITE).  Screen I/O is
CC0B:               * significantly faster when the cursor is disabled.  This
CC0B:               * feature is supported by Pascal 1.2 and later.
CC0B:               *
CC0B:AD FB 04       PASINVERT LDA   VMODE         ;Called by pascal to
CC0E:29 10                    AND   #M.CURSOR     ;display cursor
CC10:D0 0A    CC1C            BNE   INVX          ;=>cursor off, don't invert
CC12:         CC12  INVERT    EQU   *
CC12:20 1D CC                 JSR   PICKY         ;load Y and get char
CC15:48                       PHA
CC16:49 80                    EOR   #$80          ;FLIP INVERSE/NORMAL
CC18:20 B3 C3                 JSR   STORY         ;stuff onto screen
CC1B:68                       PLA                 ;for RDCHAR
CC1C:60             INVX      RTS
CC1D:               *
CC1D:               * PICK lifts a character from the screen in either
CC1D:               * 40 or 80 columns from the current cursor position.
CC1D:               * If the alternate character set is switched in,
CC1D:               * character codes $0-$1F are returned as $40-$5F (which
CC1D:               * is what must have been originally printed to the location).
CC1D:               *
CC1D:5A             PICKY     PHY                 ;save Y
CC1E:20 9D CC                 JSR   GETCUR        ;get newest cursor into Y
CC21:AD 1F C0                 LDA   RD80VID       ;80 columns?
CC24:10 17    CC3D            BPL   PICK1         ;=>no
CC26:8D 01 C0                 STA   SET80COL      ;force 80STORE if 80 columns
CC29:98                       TYA
CC2A:45 20                    EOR   WNDLFT        ;C=1 if char in main RAM
CC2C:6A                       ROR   A             ;get low bit into carry
CC2D:B0 04    CC33            BCS   PICK2         ;=>store in main memory
CC2F:AD 55 C0                 LDA   TXTPAGE2      ;else switch in page 2
CC32:C8                       INY                 ;for odd left, aux bytes
CC33:98             PICK2     TYA                 ;divide pos'n by 2
CC34:4A                       LSR   A
CC35:A8                       TAY                 ;and use as offset into line
CC36:B1 28                    LDA   (BASL),Y      ;pick character
CC38:8D 54 C0                 STA   TXTPAGE1      ;80 columns, switch in
CC3B:80 02    CC3F            BRA   PICK3         ;skip 40 column pick
CC3D:B1 28          PICK1     LDA   (BASL),Y      ;pick 40 column char
CC3F:2C 1E C0       PICK3     BIT   ALTCHARSET    ;only allow if alt set
CC42:10 06    CC4A            BPL   PICK4
CC44:C9 20                    CMP   #$20
CC46:B0 02    CC4A            BCS   PICK4
CC48:09 40                    ORA   #$40
CC4A:7A             PICK4     PLY                 ;restore real Y
CC4B:60                       RTS
CC4C:               *
CC4C:               * SHOWCUR displays either a checkerboard cursor, a solid
CC4C:               * rectangle, or the current cursor character, depending
CC4C:               * on the value of the CURSOR location.  0=inverse cursor,
CC4C:               * $FF=checkerboard cursor, anything else is displayed
CC4C:               * after being anded with inverse mask.
```

```
CC4C:                 *
CC4C:AC FB 07         SHOWCUR   LDY   CURSOR         ;what's my type?
CC4F:D0 02    CC53              BNE   NOTINV         ;=>not inverse
CC51:80 BF    CC12              BRA   INVERT         ;else invert the char (exit)
CC53:                 *
CC53:                 * Exit with char in accumulator
CC53:                 *
CC53:20 1D CC         NOTINV    JSR   PICKY          ;get char on screen
CC56:48                         PHA                  ;preserve it
CC57:8D 7B 07                   STA   NXTCUR         ;save for update
CC5A:98                         TYA                  ;test for checkerboard
CC5B:C8                         INY
CC5C:F0 0D    CC6B              BEQ   NOTINV2        ;=>checkerboard, display it
CC5E:7A                         PLY                  ;test char
CC5F:5A                         PHY
CC60:30 09    CC6B              BMI   NOTINV2        ;don't need inverse
CC62:AD 1E C0                   LDA   ALTCHARSET     ;mask = $7F if alternate
CC65:09 7F                      ORA   #$7F           ; character set,
CC67:4A                         LSR   A              ;$3F if normal char set
CC68:2D FB 07         NOTINV1   AND   CURSOR         ;form char to display
CC6B:20 B3 C3         NOTINV2   JSR   STORY          ;and display it
CC6E:68                         PLA                  ;restore real char
CC6F:60                         RTS
CC70:                 *
CC70:                 * The UPDATE routine increments the random seed.
CC70:                 * If a certain value is reached and we are in Apple II
CC70:                 * mode, the blinking check cursor is updated.  If a
CC70:                 * key has been pressed, the old char is replaced on the
CC70:                 * screen, and we return with BMI.
CC70:                 *
CC70:                 * NOTE: this routine used by COMM firmware!!
CC70:                 *
CC70:48               UPDATE    PHA                  ;save char
CC71:E6 4E                      INC   RNDL           ;update seed
CC73:D0 1C    CC91              BNE   UD2            ;check for key
CC75:A5 4F                      LDA   RNDH
CC77:E6 4F                      INC   RNDH
CC79:45 4F                      EOR   RNDH
CC7B:29 10                      AND   #$10           ;need to update cursor?
CC7D:F0 12    CC91              BEQ   UD2            ;=>no, check for key
CC7F:AD FB 07                   LDA   CURSOR         ;what cursor are we using?
CC82:F0 0D    CC91              BEQ   UD2            ;=>//e cursor, leave alone
CC84:20 1D CC                   JSR   PICKY          ;get the character into A
CC87:AC 7B 07                   LDY   NXTCUR         ;get next character
CC8A:8D 7B 07                   STA   NXTCUR         ;save next next character
CC8D:98                         TYA
CC8E:20 B3 C3                   JSR   STORY          ;and print it
CC91:68               UD2       PLA                  ;get real char
CC92:20 AD C9                   JSR   XBITKBD        ;was a key pressed?
CC95:10 28    CCBF              BPL   GETCURX        ;=>no key pressed
CC97:20 B3 C3         CLRKBD    JSR   STORY          ;restore old key
CC9A:4C 8F C9                   JMP   XRDKBD         ;look for keystroke and exit
CC9D:                 *
CC9D:                 * ON CURSORS.  Whenever the horizontal cursor position is
CC9D:                 * needed, a call to GETCUR is done. This is the equivalent
CC9D:                 * of a LDY CH.  This returns the current cursor for II and
CC9D:                 * //e mode, which may have been poked as either CH or OURCH.
CC9D:                 *
```

```
CC9D:               * It also forces CH and OLDCH to 0 if 80 column mode active.
CC9D:               * This prevents LDY CH, STA (BASL),Y from trashing non screen
CC9D:               * memory. It works just like the //e.
CC9D:               *
CC9D:               * All routines that update the cursor's horizontal position
CC9D:               * are here.  This ensures that the newest value of the cursor
CC9D:               * is always used, and that 80 column CH is always 0.
CC9D:               *
CC9D:               * GETCUR only affects the Y register
CC9D:               *
CC9D:A4 24          GETCUR    LDY   CH            ;if CH=OLDCH, then
CC9F:CC 7B 04                 CPY   OLDCH         ;OURCH is valid
CCA2:D0 03    CCA7            BNE   GETCUR1       ;=>else CH must have been changed
CCA4:AC 7B 05                 LDY   OURCH         ;use OURCH
CCA7:C4 21          GETCUR1   CPY   WNDWDTH       ;is the value too big
CCA9:90 02    CCAD            BCC   GETCUR2       ;=>no, fits just fine
CCAB:A0 00                    LDY   #0            ;else force CH to 0
CCAD:               *
CCAD:               * GETCUR2 is commonly used to set the current cursor
CCAD:               * position when Y can be used.
CCAD:               *
CCAD:8C 7B 05       GETCUR2   STY   OURCH         ;update real cursor
CCB0:2C 1F C0                 BIT   RD80VID       ;80 columns?
CCB3:10 02    CCB7            BPL   GETCUR3       ;=>no, set all cursors
CCB5:A0 00                    LDY   #0            ;yes, peg CH to 0
CCB7:84 24          GETCUR3   STY   CH
CCB9:8C 7B 04                 STY   OLDCH
CCBC:AC 7B 05                 LDY   OURCH         ;get cursor
CCBF:60             GETCURX   RTS                 ;and fly...
CCC0:                         INCLUDE ESCAPE
```

```
CCC0:                 * START AN ESCAPE SEQUENCE:
CCC0:                 *  WE HANDLE THE FOLLOWING ONES:
CCC0:                 *    @ - HOME & CLEAR
CCC0:                 *    A - Cursor right
CCC0:                 *    B - Cursor left
CCC0:                 *    C - Cursor down
CCC0:                 *    D - Cursor up
CCC0:                 *    E - CLR TO EOL
CCC0:                 *    F - CLR TO EOS
CCC0:                 *    I, Up Arrow - CURSOR UP (stay escape)
CCC0:                 *    J, Lft Arrow - CURSOR LEFT (stay escape)
CCC0:                 *    K, Rt Arrow - CURSOR RIGHT (stay escape)
CCC0:                 *    M, Dn Arrow - CURSOR DOWN  (stay escape)
CCC0:                 *    4 - GOTO 40 COLUMN MODE
CCC0:                 *    8 - GOTO 80 COLUMN MODE
CCC0:                 * CTL-D- Disable the printing of control chars
CCC0:                 * CTL-E- Enable the printing of control chars
CCC0:                 * CTL-Q- QUIT (PR#0/IN#0)
CCC0:                 *
CCC0:B9 0C CD         ESC3      LDA   ESCCHAR,Y      ;GET CHAR TO "PRINT"
CCC3:5A                         PHY                  ;save index
CCC4:20 58 CD                   JSR   CTLCHAR        ;execute character
CCC7:7A                         PLY                  ;restore index
CCC8:C0 08                      CPY   #YHI           ;If Y<YHI, stay escape
CCCA:B0 21    CCED              BCS   ESCRDKEY       ;=>exit escape mode
CCCC:                 *
CCCC:                 * This is the entry point called by RDKEY iff escapes
CCCC:                 * are enabled and an escape is encountered.  The next
CCCC:                 * keypress is read and processed.  If it is a key that
CCCC:                 * terminates escape mode, a new key is read by ESCRDKEY.
CCCC:                 * If escape mode should not be terminated, NEWESC is
CCCC:                 * called again.
CCCC:                 *
CCCC:20 1D CC         NEWESC    JSR   PICKY          ;get current character
CCCF:48                         PHA                  ;and save it
CCD0:29 80                      AND   #$80           ;save invert bit
CCD2:49 AB                      EOR   #$AB           ;make it inverted "+"
CCD4:20 B3 C3                   JSR   STORY          ;and pop it on the screen
CCD7:20 AD C9         ESC0      JSR   XBITKBD        ;check for keystroke
CCDA:10 FB    CCD7              BPL   ESC0
CCDC:68                         PLA                  ;get old char
CCDD:20 97 CC                   JSR   CLRKBD         ;restore char, get key
CCE0:20 9B C3                   JSR   UPSHIFT        ;upshift esc char
CCE3:A0 13            ESC1      LDY   #ESCNUM        ;COUNT/INDEX
CCE5:D9 F8 CC         ESC2      CMP   ESCTAB,Y       ;IS IT A VALID ESCAPE?
CCE8:F0 D6    CCC0              BEQ   ESC3           =>yes
CCEA:88                         DEY
CCEB:10 F8    CCE5              BPL   ESC2           ;TRY 'EM ALL...
CCED:                 *
CCED:                 * End of escape sequence, read next character.
CCED:                 * This is initially called by RDCHAR which is usually called
CCED:                 * by GETLN to read characters with escapes enabled.
CCED:                 *
CCED:A9 08            ESCRDKEY  LDA   #M.CTL         ;enable escape sequences
CCEF:1C FB 04                   TRB   VMODE
CCF2:20 0C FD                   JSR   RDKEY          ;read char with escapes
CCF5:4C 44 FD                   JMP   NOESCAPE       ;got the key, disable escapes
CCF8:                 *
```

```
CCF8:                 * When in escape mode, the characters in ESCTAB (high)
CCF8:                 * bits set), are mapped into the characters in ESCCHAR.
CCF8:                 * These characters are then executed by a call to CTLCHAR.
CCF8:                 *
CCF8:                 * CTLCHAR looks up a character in the table starting at
CCF8:                 * CTLTAB.  It uses the current index as an index into the
CCF8:                 * table of routine addresses, CTLADR.  If the character is
CCF8:                 * not in the table, a call to VIDOUT1 is done in case the
CCF8:                 * character is BS, LF, CR, or BEL.
CCF8:                 *
CCF8:                 * NOTE: CTLON and CTLOFF are not accessible except through
CCF8:                 * and escape sequence
CCF8:                 *
CCF8:                            MSB   ON              ;high bit on
CCF8:         CCF8    ESCTAB     EQU   *
CCF8:CA                          ASC   'J'             ;left (stay esc)
CCF9:88                          DFB   $88             ;left arrow (stay esc)
CCFA:CD                          ASC   'M'             ;down (stay esc)
CCFB:8B                          DFB   $8B             ;up arrow (stay esc)
CCFC:95                          DFB   $95             ;right arrow (stay esc)
CCFD:8A                          DFB   $8A             ;down arrow (stay esc)
CCFE:C9                          ASC   'I'             ;up (stay esc)
CCFF:CB                          ASC   'K'             ;right (stay esc)
CD00:         0008    YHI        EQU   *-ESCTAB
CD00:C2                          ASC   'B'             ;left
CD01:C3                          ASC   'C'             ;down
CD02:C4                          ASC   'D'             ;up
CD03:C1                          ASC   'A'             ;right
CD04:C0                          ASC   '@'             ;formfeed
CD05:C5                          ASC   'E'             ;clear EOL
CD06:C6                          ASC   'F'             ;clear EOS
CD07:B4                          ASC   '4'             ;40 column mode
CD08:B8                          ASC   '8'             ;80 column mode
CD09:91                          DFB   $91             ;CTL-Q = QUIT
CD0A:84                          DFB   $84             ;CTL-D ;ctl char disable
CD0B:85                          DFB   $85             ;CTL-E ;ctl char enable
CD0C:                 *
CD0C:         0013    ESCNUM     EQU   *-ESCTAB-1
CD0C:                 *
CD0C:         CD0C    ESCCHAR    EQU   *               ;list of escape chars
CD0C:88                          DFB   $88             ;J: BS (stay esc)
CD0D:88                          DFB   $88             ;<-:BS (stay esc)
CD0E:8A                          DFB   $8A             ;M: LF (stay esc)
CD0F:9F                          DFB   $9F             ;UP:US (stay esc)
CD10:9C                          DFB   $9C             ;->:FS (stay esc)
CD11:8A                          DFB   $8A             ;DN: LF (stay esc)
CD12:9F                          DFB   $9F             ;I: UP (stay esc)
CD13:9C                          DFB   $9C             ;K: RT (stay esc)
CD14:88                          DFB   $88             ;ESC-B = BS
CD15:         CD15    CTLTAB     EQU   *               ;list of control characters
CD15:8A                          DFB   $8A             ;ESC-C = DN
CD16:9F                          DFB   $9F             ;ESC-D = UP
CD17:9C                          DFB   $9C             ;ESC-A = RT
CD18:8C                          DFB   $8C             ;@: Formfeed
CD19:9D                          DFB   $9D             ;E: CLREOL
CD1A:8B                          DFB   $8B             ;F: CLREOP
CD1B:91                          DFB   $91             ;SET40
CD1C:92                          DFB   $92             ;SET80
```

```
CD1D:95                           DFB   $95           ;QUIT
CD1E:04                           DFB   $04           ;Disable controls (escape only)
CD1F:05                           DFB   $05           ;Enable controls (escape only)
CD20:                 * escape chars end here
CD20:85                           DFB   $85           ;X.CUR.ON
CD21:86                           DFB   $86           ;X.CUR.OFF
CD22:8E                           DFB   $8E           ;Normal
CD23:8F                           DFB   $8F           ;Inverse
CD24:96                           DFB   $96           ;Scroll down
CD25:97                           DFB   $97           ;Scroll up
CD26:98                           DFB   $98           ;mouse chars off
CD27:99                           DFB   $99           ;home cursor
CD28:9A                           DFB   $9A           ;clear line
CD29:9B                           DFB   $9B           ;mouse chars on
CD2A:                 *
CD2A:           0014  CTLNUM      EQU   *-CTLTAB-1
CD2A:                 *
CD2A:           CD2A  CTLADR      EQU   *
CD2A:66 FC                        DW    LF            ;move cursor down
CD2C:1A FC                        DW    UP            ;move cursor up
CD2E:A0 FB                        DW    NEWADV        ;forward a space
CD30:58 FC                        DW    HOME          ;home cursor, clear screen
CD32:9C FC                        DW    CLREOL        ;clear to end of line
CD34:42 FC                        DW    CLREOP        ;clear to end of page
CD36:C0 CD                        DW    SET40         ;set 40 column mode
CD38:BE CD                        DW    SET80         ;set 80 column mode
CD3A:45 CE                        DW    QUIT          ;Quit video firmware
CD3C:91 CD                        DW    CTLOFF        ;disable //e control chars
CD3E:95 CD                        DW    CTLON         ;enable //e control chars
CD40:89 CD                        DW    X.CUR.ON      ;turn on cursor (pascal)
CD42:8D CD                        DW    X.CUR.OFF     ;turn off cursor (pascal)
CD44:B0 CD                        DW    X.SO          ;normal video
CD46:B7 CD                        DW    X.SI          ;inverse video
CD48:30 CB                        DW    SCROLLDN      ;scroll down a line
CD4A:35 CB                        DW    SCROLLUP      ;scroll up a line
CD4C:9F CD                        DW    MOUSOFF       ;disable mouse characters
CD4E:A5 CD                        DW    HOMECUR       ;move cursor home
CD50:A0 FC                        DW    CLRLIN        ;clear current line
CD52:99 CD                        DW    MOUSON        ;enable mouse characters
CD54:                 *
CD54:                             MSB   ON
CD54:                 *
CD54:                 * CTLCHAR executes the control character in the
CD54:                 * accumulator.  If it is called by Pascal, the character
CD54:                 * is always executed.  If it is called by the video
CD54:                 * firmware, the character is executed if M.CTL is set
CD54:                 * and M.CTL2 is clear.
CD54:                 *
CD54:                 * Note: This routine is only called if the video firmware
CD54:                 * is active. The Monitor ROM calls VIDOUT1 if the video
CD54:                 * firmware is inactive.
CD54:                 *
CD54:2C C1 CB         CTLCHAR0    BIT   SEV1          ;set V (use M.CTL)
CD57:50                           DFB   $50           ;BVC opcode (never taken)
CD58:                 *
CD58:B8               CTLCHAR     CLV                 ;Always do control character
CD59:DA                           PHX                 ;save X
CD5A:8D F8 04                     STA   TEMP1         ;temp save of A
```

```
CD5D:20 04 FC                   JSR   VIDOUT1        ;try to execute CR, LF, BS, or BEL
CD60:CD F8 04                   CMP   TEMP1          ;if acc has changed
CD63:D0 0A    CD6F              BNE   CTLDONE        ;then function done
CD65:A2 14                      LDX   #CTLNUM        ;number of CTL chars
CD67:DD 15 CD       FNDCTL      CMP   CTLTAB,X       ;is it in table
CD6A:F0 05    CD71              BEQ   CTLGO          ;=>yes, should we execute?
CD6C:CA                         DEX                  ;else check next
CD6D:10 F8    CD67              BPL   FNDCTL         ;=>try next one
CD6F:FA             CTLDONE     PLX                  ;restore X
CD70:60                         RTS                  ;and return
CD71:               *
CD71:48             CTLGO       PHA                  ;save A
CD72:50 0C    CD80              BVC   CTLGO1         ;V clear, always do (pascal,escape)
CD74:AD FB 04                   LDA   VMODE          ;controls are enabled iff
CD77:29 28                      AND   #M.CTL+M.CTL2  ; M.CTL = 1 and
CD79:49 08                      EOR   #M.CTL         ; M.CTL2 = 0
CD7B:F0 03    CD80              BEQ   CTLGO1         ;=>they're enabled!!
CD7D:68             CGO         PLA                  ;restore A
CD7E:FA                         PLX                  ;restore X
CD7F:60                         RTS                  ;and return
CD80:               *
CD80:8A             CTLGO1      TXA                  ;double X as index
CD81:0A                         ASL   A              ;into address table
CD82:AA                         TAX
CD83:68                         PLA                  ;restore A
CD84:20 A4 FC                   JSR   CTLDO          ;execute the char
CD87:FA                         PLX                  ;restore X
CD88:60                         RTS                  ;and return
CD89:               *
CD89:               * X.CUR.ON = Allow Pascal cursor display
CD89:               * X.CUR.OFF = Disable Pascal cursor display
CD89:               * Cursor is not displayed during call, so it will
CD89:               * be right when "redisplayed".
CD89:               * Note: Though these commands are executed from BASIC,
CD89:               * they have no effect on firmware operation.
CD89:               *
CD89:A9 10          X.CUR.ON    LDA   #M.CURSOR      ;clear cursor bit
CD8B:80 0E    CD9B              BRA   CLRIT
CD8D:               *
CD8D:A9 10          X.CUR.OFF   LDA   #M.CURSOR      ;set cursor bit
CD8F:80 10    CDA1              BRA   SETIT
CD91:               *
CD91:               * The control characters other than CR,LF,BEL,BS
CD91:               * are normally enabled when video firmware is active.
CD91:               * They can be disabled and enabled using the ESC-D
CD91:               * and ESC-E escape sequences.
CD91:               *
CD91:A9 20          CTLOFF      LDA   #M.CTL2        ;disable control characters
CD93:80 0C    CDA1              BRA   SETIT          ;by setting M.CTL2
CD95:               *
CD95:A9 20          CTLON       LDA   #M.CTL2        ;enable control characters
CD97:80 02    CD9B              BRA   CLRIT          ;by clearing M.CTL2
CD99:               *
CD99:               * Enable mouse text by clearing M.MOUSE
CD99:               *
CD99:A9 01          MOUSON      LDA   #M.MOUSE
CD9B:1C FB 04       CLRIT       TRB   VMODE
CD9E:60                         RTS
```

```
CD9F:                 *
CD9F:                 * Disable mouse text by setting M.MOUSE
CD9F:                 *
CD9F:A9 01            MOUSOFF   LDA   #M.MOUSE
CDA1:0C FB 04         SETIT     TSB   VMODE
CDA4:60                         RTS
CDA5:                 *
CDA5:                 * EXECUTE HOME:
CDA5:                 *
CDA5:20 E9 FE         HOMECUR   JSR   CLRCH         ;move cursors to far left
CDA8:A8                         TAY                 ;(probably not needed)
CDA9:A5 22                      LDA   WNDTOP        ;and to top of window
CDAB:85 25                      STA   CV
CDAD:4C 88 FC                   JMP   NEWVTABZ      ;then set base address, OURCV
CDB0:                 *
CDB0:                 * EXECUTE "NORMAL VIDEO"
CDB0:                 *
CDB0:20 84 FE         X.SO      JSR   SETNORM       ;set INVFLG to $FF
CDB3:A9 04                      LDA   #M.VMODE      ;then clear inverse mode bit
CDB5:80 E4    CD9B              BRA   CLRIT
CDB7:                 *
CDB7:                 * EXECUTE "INVERSE VIDEO"
CDB7:                 *
CDB7:20 80 FE         X.SI      JSR   SETINV        ;set INVFLG to $3F
CDBA:A9 04                      LDA   #M.VMODE      ;then set inverse mode bit
CDBC:80 E3    CDA1              BRA   SETIT
CDBE:                 *
CDBE:                 * EXECUTE '40COL MODE' or '80COL MODE':
CDBE:                 *
CDBE:38               SET80     SEC                 ;flag an 80 column window
CDBF:90                         DFB   $90           ;BCC opcode (never taken)
CDC0:18               SET40     CLC                 ;flag a 40 column window
CDC1:2C FB 04                   BIT   VMODE         ;but...is it pascal?
CDC4:10 54    CE1A              BPL   SETX          ;=>yes, don't execute
CDC6:08                         PHP                 ;save window size
CDC7:20 1B CE                   JSR   HOOKITUP      ;COPYROM if needed, set I/O hooks
CDCA:28                         PLP                 ;and get 40/80
CDCB:80 08    CDD5              BRA   WIN0          ;=>set window
CDCD:                 *
CDCD:                 * CHK80 is called by PR#0 to convert to 40 if it was
CDCD:                 * 80. Otherwise the window is left ajar.
CDCD:                 *
CDCD:2C 1F C0         CHK80     BIT   RD80VID       ;don't set 40 if
CDD0:10 48    CE1A              BPL   SETX          ;already 40
CDD2:                 *
CDD2:18               WIN40     CLC                 ;flag 40 column window
CDD3:B0                         DFB   $B0           ;BCS opcode (never taken)
CDD4:38               WIN80     SEC                 ;flag 80 column window
CDD5:64 22            WIN0      STZ   WNDTOP        ;set window top now
CDD7:2C 1A C0                   BIT   RDTEXT        ;for text or mixed
CDDA:30 04    CDE0              BMI   WIN1          ;=>text
CDDC:A9 14                      LDA   #20
CDDE:85 22                      STA   WNDTOP        ;used by 80<->40 conversion
CDE0:2C 1F C0         WIN1      BIT   RD80VID       ;80 columns now?
CDE3:08                         PHP                 ;save 80 or 40
CDE4:B0 07    CDED              BCS   WIN2          ;=>80: convert if 40
CDE6:10 0A    CDF2              BPL   WIN3          ;=>40: no convert
CDE8:20 53 CE                   JSR   SCRN84        ;80: convert to 40
```

```
CDEB:80 05    CDF2             BRA   WIN3         ;done converting
CDED:30 03    CDF2  WIN2       BMI   WIN3         ;=>80: no convert
CDEF:20 80 CE                  JSR   SCRN48       ;40: convert to 80
CDF2:20 9D CC       WIN3       JSR   GETCUR       ;determine absolute CH
CDF5:98                        TYA                ;in case the window setting
CDF6:18                        CLC                ;was different
CDF7:65 20                     ADC   WNDLFT
CDF9:28                        PLP                ;pin to right edge if
CDFA:B0 06    CE02             BCS   WIN4         ;80 to 40 leaves cursor
CDFC:C9 28                     CMP   #40          ;off the screen
CDFE:90 02    CE02             BCC   WIN4
CE00:A9 27                     LDA   #39
CE02:20 EC FE       WIN4       JSR   SETCUR       ;set new cursor
CE05:A5 25                     LDA   CV           ;set new base address
CE07:20 C1 FB                  JSR   BASCALC      ;for left = 0 (always)
CE0A:               *
CE0A:64 20          WNDREST    STZ   WNDLFT       ;Called by INIT and Pascal
CE0C:A9 18                     LDA   #$18         ;and bottom
CE0E:85 23                     STA   WNDBTM
CE10:A9 28                     LDA   #$28         ;set left,width,bottom
CE12:2C 1F C0                  BIT   RD80VID      ;set width to 80 if 80 columns
CE15:10 01    CE18             BPL   WIN5
CE17:0A                        ASL   A
CE18:85 21          WIN5       STA   WNDWDTH      ;set width
CE1A:60             SETX       RTS                ;exit used by SET40/80
CE1B:               *
CE1B:               * Turn on video firmware:
CE1B:               *
CE1B:               * This routine is used by BASIC init, ESC-4, ESC-8
CE1B:               * It copies the Monitor ROM to the language card
CE1B:               * if necessary; it sets the input and output hooks to
CE1B:               * $C30x; it sets all switches for video firmware operation
CE1B:               *
CE1B:2C 7B 06       HOOKITUP   BIT   VFACTV       ;don't touch hooks
CE1E:10 11    CE31             BPL   VIDMODE      ;if video firmware already active
CE20:20 38 C3       HOOKUP     JSR   COPYROM      ;Copy ROM to LC?
CE23:A9 05          SETHOOKS   LDA   #>C3KEYIN    ;set up $C300 hooks
CE25:85 38                     STA   KSWL
CE27:A9 07                     LDA   #>C3COUT1
CE29:85 36                     STA   CSWL
CE2B:A9 C3                     LDA   #<C3COUT1
CE2D:85 39                     STA   KSWH
CE2F:85 37                     STA   CSWH
CE31:               *
CE31:               * Now set the video firmware active
CE31:               *
CE31:9C FB 07       VIDMODE    STZ   CURSOR       ;set a solid inverse cursor
CE34:A9 08                     LDA   #M.CTL       ;preserve M.CTL bit
CE36:2D FB 04                  AND   VMODE
CE39:09 81                     ORA   #M.PASCAL+M.MOUSE ;no pascal,mouse
CE3B:               *
CE3B:               * Pascal calls here to set its mode
CE3B:               *
CE3B:8D FB 04       PVMODE     STA   VMODE        ;set mode bits
CE3E:9C 7B 06                  STZ   VFACTV       ;say video firmware active
CE41:8D 0F C0                  STA   SETALTCHAR   ;and set alternate char set
CE44:60             QX         RTS
CE45:               *
```

```
CE45:                * QUIT converts the screen from 80 to 40 if necessary,
CE45:                * sets a 40 column window, and restores the normal I/O
CE45:                * hooks (COUTl and KEYIN).
CE45:                *
CE45:2C FB 04          QUIT      BIT   VMODE         ;no quitting from pascal
CE48:10 FA    CE44               BPL   QX
CE4A:20 D2 CD                    JSR   WIN40         ;first, do an escape 4
CE4D:20 89 FE          ZZQUIT    JSR   SETKBD        ;do a IN#0  (used by COMM)
CE50:4C 93 FE                    JMP   SETVID        ;and a PR#0
```

```
CE53:                *
CE53:                * SCRN84 and SCRN48 convert screens between 40 & 80 cols.
CE53:                * WNDTOP must be set up to indicate the last line to
CE53:                * be done.  All registers are trashed.
CE53:                *
CE53:A2 17           SCRN84   LDX   #23          ;start at bottom of screen
CE55:8D 01 C0                 STA   SET80COL     ;allow page 2 access
CE58:8A              SCR1     TXA                ;calc base for line
CE59:20 C1 FB                 JSR   BASCALC
CE5C:A0 27                    LDY   #39          ;start at right of screen
CE5E:5A              SCR2     PHY                ;save 40 index
CE5F:98                       TYA                ;div by 2 for 80 column index
CE60:4A                       LSR   A
CE61:B0 03    CE66            BCS   SCR3
CE63:2C 55 C0                 BIT   TXTPAGE2     ;even column, do page 2
CE66:A8              SCR3     TAY                ;get 80 index
CE67:B1 28                    LDA   (BASL),Y     ;get 80 char
CE69:2C 54 C0                 BIT   TXTPAGE1     ;restore page1
CE6C:7A                       PLY                ;get 40 index
CE6D:91 28                    STA   (BASL),Y
CE6F:88                       DEY
CE70:10 EC    CE5E            BPL   SCR2         ;do next 40 byte
CE72:CA                       DEX                ;do next line
CE73:30 04    CE79            BMI   SCR4         ;=>done with setup
CE75:E4 22                    CPX   WNDTOP       ;at top yet?
CE77:B0 DF    CE58            BCS   SCR1
CE79:8D 00 C0        SCR4     STA   CLR80COL     ;clear 80STORE for 40 columns
CE7C:8D 0C C0                 STA   CLR80VID     ;clear 80VID for 40 columns
CE7F:60                       RTS
CE80:                *
CE80:A2 17           SCRN48   LDX   #23          ;start at bottom of screen
CE82:8A              SCR5     TXA                ;set base for current line
CE83:20 C1 FB                 JSR   BASCALC
CE86:A0 00                    LDY   #0           ;start at left of screen
CE88:8D 01 C0                 STA   SET80COL     ;enable page2 store
CE8B:B1 28           SCR6     LDA   (BASL),Y     ;get 40 column char
CE8D:5A              SCR8     PHY                ;save 40 column index
CE8E:48                       PHA                ;save char
CE8F:98                       TYA                ;div 2 for 80 column index
CE90:4A                       LSR   A
CE91:B0 03    CE96            BCS   SCR7         ;save on page1
CE93:8D 55 C0                 STA   TXTPAGE2
CE96:A8              SCR7     TAY                ;get 80 column index
CE97:68                       PLA                ;now save character
CE98:91 28                    STA   (BASL),Y
CE9A:8D 54 C0                 STA   TXTPAGE1     ;flip page1
CE9D:7A                       PLY                ;restore 40 column index
CE9E:C8                       INY                ;move to the right
CE9F:C0 28                    CPY   #40          ;at right yet?
CEA1:90 E8    CE8B            BCC   SCR6         ;=>no, do next column
CEA3:20 CF CB                 JSR   CLRHALF      ;clear half of screen
CEA6:CA                       DEX                ;else do next line of screen
CEA7:30 04    CEAD            BMI   SCR9         ;=>done with top line
CEA9:E4 22                    CPX   WNDTOP       ;at top yet?
CEAB:B0 D5    CE82            BCS   SCR5
CEAD:8D 0D C0        SCR9     STA   SET80VID     ;convert to 80 columns
CEB0:60                       RTS
CEB1:                         INCLUDE PASCAL     ;Pascal support stuff
```

```
CEB1:AA                 PSTATUS  TAX                ;is request code = 0?
CEB2:F0 08    CEBC               BEQ    PIORDY      ;=>yes, ready for output
CEB4:CA                          DEX                ;check for any input
CEB5:D0 07    CEBE               BNE    PSTERR      ;=>bad request, return error
CEB7:20 AD C9                    JSR    XBITKBD     ;test keyboard
CEBA:10 04    CEC0               BPL    PNOTRDY     ;=>no keystroked
CEBC:38                 PIORDY   SEC                ;good return
CEBD:60                          RTS
CEBE:A2 03              PSTERR   LDX    #3          ;else flag error
CEC0:18                 PNOTRDY  CLC
CEC1:60                          RTS
CEC2:                   *
CEC2:                   * PASCAL OUTPUT:
CEC2:                   *
CEC2:         CEC2      PWRITE   EQU    *
CEC2:09 80                       ORA    #$80        ;turn on high bit
CEC4:AA                          TAX                ;save character
CEC5:20 54 CF                    JSR    PSETUP2     ;SETUP ZP STUFF, don't set ROM
CEC8:A9 08                       LDA    #M.GOXY     ;ARE WE DOING GOTOXY?
CECA:2C FB 04                    BIT    VMODE
CECD:D0 2B    CEFA               BNE    GETX        ;=>Doing X or Y?
CECF:8A                          TXA                ;now check for control char
CED0:89 60                       BIT    #$60        ;is it control?
CED2:F0 45    CF19               BEQ    PCTL        ;=>yes, do control
CED4:AC 7B 05                    LDY    OURCH       ;get horizontal position
CED7:24 32                       BIT    INVFLG      ;check for inverse
CED9:30 02    CEDD               BMI    PWR1        ;normal, go store it
CEDB:29 7F                       AND    #$7F
CEDD:20 C1 C3           PWR1     JSR    STORE       ;now store it (erasing cursor)
CEE0:C8                          INY                ;INC CH
```

```
CEE1:8C 7B 05                  STY   OURCH
CEE4:C4 21                     CPY   WNDWDTH
CEE6:90 0C     CEF4            BCC   PWRET
CEE8:20 60 C3                  JSR   SETROM
CEEB:20 E9 FE                  JSR   CLRCH       ;set cursor position to 0
CEEE:20 66 FC                  JSR   LF
CEF1:20 54 C3       PWRITERET  JSR   RESETLC
CEF4:20 0B CC       PWRET      JSR   PASINVERT   ;display new cursor
CEF7:A2 00          PRET       LDX   #$0         ;return with no error
CEF9:60                        RTS
CEFA:               *
CEFA:               *  HANDLE GOTOXY STUFF:
CEFA:               *
CEFA:          CEFA GETX       EQU   *
CEFA:20 0B CC                  JSR   PASINVERT   ;turn off cursor
CEFD:8A                        TXA               ;get character
CEFE:38                        SEC
CEFF:E9 A0                     SBC   #160        ;MAKE BINARY
CF01:2C FB 06                  BIT   XCOORD      ;doing X?
CF04:30 2A     CF30            BMI   PSETX       ;=>yes, set it
CF06:               *
CF06:               * Set Y and do the GOTOXY
CF06:               *
CF06:          CF06 GETY       EQU   *
CF06:8D FB 05                  STA   OURCV
CF09:20 71 CF                  JSR   PASCALC     ;calc base addr
CF0C:AC FB 06                  LDY   XCOORD
CF0F:20 AD CC                  JSR   GETCUR2     ;set proper cursors
```

```
CF12:A9 08                           LDA   #M.GOXY      ;turn off gotoxy
CF14:1C FB 04                        TRB   VMODE
CF17:80 DB    CEF4                   BRA   PWRET        ;=>DONE (ALWAYS TAKEN)
CF19:                      *
CF19:20 0B CC              PCTL      JSR   PASINVERT    ;turn off cursor
CF1C:8A                              TXA                ;get char
CF1D:C9 9E                           CMP   #$9E         ;is it gotoXY?
CF1F:F0 08    CF29                   BEQ   STARTXY      ;=>yes, start it up
CF21:20 60 C3                        JSR   SETROM       ;must switch in ROM for controls
CF24:20 58 CD                        JSR   CTLCHAR      ;EXECUTE IT IF POSSIBLE
CF27:80 C8    CEF1                   BRA   PWRITERET    ;=>display new cursor, exit
CF29:                      *
CF29:                      * START THE GOTOXY SEQUENCE:
CF29:                      *
CF29:         CF29         STARTXY   EQU   *
CF29:A9 08                           LDA   #M.GOXY
CF2B:0C FB 04                        TSB   VMODE        ;turn on gotoxy
CF2E:A9 FF                           LDA   #$FF         ;set XCOORD to -1
CF30:8D FB 06              PSETX     STA   XCOORD       ;set X
CF33:80 BF    CEF4                   BRA   PWRET        ;=>display cursor and exit
CF35:                      *
CF35:                      * PASCAL INPUT:
CF35:                      *
CF35:20 54 CF              PASREAD   JSR   PSETUP2      ;SETUP ZP STUFF
CF38:20 8F C9              GKEY      JSR   XRDKBD       ;key pressed?
CF3B:10 FB    CF38                   BPL   GKEY         ;=>not yet
CF3D:29 7F                           AND   #$7F         ;DROP HI BIT
CF3F:80 B6    CEF7                   BRA   PRET         ;good exit
CF41:                      *
CF41:                      * PASCAL INITIALIZATION:
CF41:                      *
CF41:         CF41         PINIT     EQU   *
CF41:A9 01                           LDA   #M.MOUSE     ;Set mode to pascal
CF43:20 3B CE                        JSR   PVMODE       ;without mouse characters
CF46:20 51 CF                        JSR   PSETUP       ;setup zero page for pascal
CF49:20 D4 CD                        JSR   WIN80        ;do 40->80 convert
CF4C:20 58 FC                        JSR   HOME         ;home and clear screen
CF4F:80 A0    CEF1                   BRA   PWRITERET    ;display cursor, set OURCH,OURCV...
CF51:                      *
CF51:         CF51         PSETUP    EQU   *
CF51:20 60 C3                        JSR   SETROM       ;save LC state, set ROM read
CF54:64 22                 PSETUP2   STZ   WNDTOP       ;set top to 0
CF56:20 0A CE                        JSR   WNDREST      ;init either 40 or 80 window
CF59:A9 FF                           LDA   #$FF         ;assume normal text
CF5B:85 32                           STA   INVFLG
CF5D:A9 04                           LDA   #M.VMODE     ;is it
CF5F:2C FB 04                        BIT   VMODE
CF62:F0 02    CF66                   BEQ   PS1          ;=>yes
CF64:46 32                           LSR   INVFLG       ;no, make flag inverse
CF66:AC 7B 05              PS1       LDY   OURCH
CF69:20 AD CC                        JSR   GETCUR2      ;set all cursors
CF6C:AD FB 05                        LDA   OURCV
CF6F:85 25                           STA   CV
CF71:                      *
CF71:                      * Put BASCALC here so we don't have to switch
CF71:                      * in the ROMs for each character output.
CF71:                      *
CF71:0A                    PASCALC   ASL   A
```

```
CF72:A8                 TAY                 ;calc base addr in BASL,H
CF73:4A                 LSR   A             ;for given line no.
CF74:4A                 LSR   A
CF75:29 03              AND   #$03          ; 0<=line no.<=$17
CF77:09 04              ORA   #$4           ; arg=000ABCDE, generate
CF79:85 29              STA   BASH          ; BASH=000001CD
CF7B:98                 TYA                 ;and
CF7C:6A                 ROR   A             ; BASL=EABAB000
CF7D:29 98              AND   #$98
CF7F:85 28     PASCLC2  STA   BASL
CF81:0A                 ASL   A
CF82:0A                 ASL   A
CF83:04 28              TSB   BASL
CF85:60                 RTS
CF86:                   INCLUDE AUXSTUFF    ;Aux RAM routines
```

```
CF86:                 *****************************************
CF86:                 * NAME     : MOVEAUX
CF86:                 * FUNCTION: PERFORM CROSSBANK MEMORY MOVE
CF86:                 * INPUT    : A1=SOURCE ADDRESS
CF86:                 *          : A2=SOURCE END
CF86:                 *          : A4=DESTINATION START
CF86:                 *          : CARRY SET=MAIN-->CARD
CF86:                 *                  CLR=CARD-->MAIN
CF86:                 * OUTPUT   : NONE
CF86:                 * VOLATILE: NOTHING
CF86:                 * CALLS    : NOTHING
CF86:                 *****************************************
CF86:           CF86  MOVEAUX   EQU   *
CF86:48                         PHA                    ;SAVE AC
CF87:AD 13 C0                   LDA   RDRAMRD          ;SAVE STATE OF
CF8A:48                         PHA                    ; MEMORY FLAGS
CF8B:AD 14 C0                   LDA   RDRAMWRT
CF8E:48                         PHA
CF8F:                 *
CF8F:                 * SET FLAGS FOR CROSSBANK MOVE:
CF8F:                 *
CF8F:90 08      CF99            BCC   MOVEC2M          ;=>CARD-->MAIN
CF91:8D 02 C0                   STA   RDMAINRAM        ;SET FOR MAIN
CF94:8D 05 C0                   STA   WRCARDRAM        ; TO CARD
CF97:B0 06      CF9F            BCS   MOVESTRT         ;=>(ALWAYS TAKEN)
CF99:                 *
CF99:           CF99  MOVEC2M   EQU   *
CF99:8D 04 C0                   STA   WRMAINRAM        ;SET FOR CARD
CF9C:8D 03 C0                   STA   RDCARDRAM        ; TO MAIN
CF9F:                 *
CF9F:           CF9F  MOVESTRT  EQU   *
CF9F:B2 3C            MOVELOOP  LDA   (A1L)            ;get a byte
CFA1:92 42                      STA   (A4L)            ;move it
CFA3:E6 42                      INC   A4L
CFA5:D0 02      CFA9            BNE   NEXTA1
CFA7:E6 43                      INC   A4H
CFA9:A5 3C            NEXTA1    LDA   A1L
CFAB:C5 3E                      CMP   A2L
CFAD:A5 3D                      LDA   A1H
CFAF:E5 3F                      SBC   A2H
CFB1:E6 3C                      INC   A1L
CFB3:D0 02      CFB7            BNE   C01
CFB5:E6 3D                      INC   A1H
CFB7:90 E6      CF9F  C01       BCC   MOVELOOP         ;=>more to move
CFB9:                 *
CFB9:8D 04 C0                   STA   WRMAINRAM        ;CLEAR FLAG2
CFBC:68                         PLA                    ;GET ORIGINAL STATE
CFBD:10 03      CFC2            BPL   C03              ;=>IT WAS OFF
CFBF:8D 05 C0                   STA   WRCARDRAM
CFC2:           CFC2  C03       EQU   *
CFC2:8D 02 C0                   STA   RDMAINRAM        ;CLEAR FLAG1
CFC5:68                         PLA                    ;GET ORIGINAL STATE
CFC6:10 03      CFCB            BPL   MOVERET          ;=>IT WAS OFF
CFC8:8D 03 C0                   STA   RDCARDRAM
CFCB:           CFCB  MOVERET   EQU   *
CFCB:68                         PLA                    ;Restore AC
CFCC:60                         RTS
```

```
CFCD:                 62 ****************************************
CFCD:                 63 * NAME     : XFER
CFCD:                 64 * FUNCTION: TRANSFER CONTROL CROSSBANK
CFCD:                 65 * INPUT    : $03ED=TRANSFER ADDR
CFCD:                 66 *          : CARRY SET=XFER TO CARD
CFCD:                 67 *                  CLR=XFER TO MAIN
CFCD:                 68 *          : VFLAG CLR=USE STD ZP/STK
CFCD:                 69 *          :       SET=USE ALT ZP/STK
CFCD:                 70 * OUTPUT   : NONE
CFCD:                 71 * VOLATILE: $03ED/03EE IN DEST BANK
CFCD:                 72 * CALLS    : NOTHING
CFCD:                 73 * NOTE     : ENTERED VIA JMP, NOT JSR
CFCD:                 74 ****************************************
CFCD:                 75 *
CFCD:          CFCD   76 XFER      EQU   *
CFCD:48               77           PHA                  ;SAVE AC ON CURRENT STACK
CFCE:                 78 *
CFCE:                 79 * COPY DESTINATION ADDRESS TO THE
CFCE:                 80 *  OTHER BANK SO THAT WE HAVE IT
CFCE:                 81 *  IN CASE WE DO A SWAP:
CFCE:                 82 *
CFCE:AD ED 03         83           LDA   $03ED          ;GET XFERADDR LO
CFD1:48               84           PHA                  ;SAVE ON CURRENT STACK
CFD2:AD EE 03         85           LDA   $03EE          ;GET XFERADDR HI
CFD5:48               86           PHA                  ;SAVE IT TOO
CFD6:                 87 *
CFD6:                 88 * SWITCH TO APPROPRIATE BANK:
CFD6:                 89 *
CFD6:90 08    CFE0    90           BCC   XFERC2M        ;=>CARD-->MAIN
CFD8:8D 03 C0         91           STA   RDCARDRAM      ;SET FOR RUNNING
CFDB:8D 05 C0         92           STA   WRCARDRAM      ; IN CARD RAM
CFDE:B0 06    CFE6    93           BCS   XFERZP         ;=> always taken
CFE0:         CFE0    94 XFERC2M   EQU   *
CFE0:8D 02 C0         95           STA   RDMAINRAM      ;SET FOR RUNNING
CFE3:8D 04 C0         96           STA   WRMAINRAM      ; IN MAIN RAM
CFE6:                 97 *
CFE6:         CFE6    98 XFERZP    EQU   *              ;SWITCH TO ALT ZP/STK
CFE6:68               99           PLA                  ;STUFF XFERADDR
CFE7:8D EE 03        100           STA   $03EE          ; HI AND
CFEA:68              101           PLA
CFEB:8D ED 03        102           STA   $03ED          ;  LO
CFEE:68              103           PLA                  ;RESTORE AC
CFEF:70 05    CFF6   104           BVS   XFERAZP        ;=>switch in alternate zp
CFF1:8D 08 C0        105           STA   SETSTDZP       ;else force standard zp
CFF4:50 03    CFF9   106           BVC   JMPDEST        ;=>always perform transfer
CFF6:8D 09 C0        107 XFERAZP   STA   SETALTZP       ;switch in alternate zp
CFF9:6C ED 03        108 JMPDEST   JMP   ($03ED)        ;=>off we go
CFFC:                109 ****************************************
CFFC:         0004   110           DS    $D000-*,$00
----- NEXT OBJECT FILE NAME IS FIRM.1
F800:         F800    31           ORG   F8ORG
F800:                 32           INCLUDE AUTOST1      ;F8 monitor rom
```

```
F800:4A               PLOT      LSR  A           ;Y-COORD/2
F801:08                         PHP              ;SAVE LSB IN CARRY
F802:20 47 F8                   JSR  GBASCALC    ;CALC BASE ADR IN GBASL,H
F805:28                         PLP              ;RESTORE LSB FROM CARRY
F806:A9 0F                      LDA  #$0F        ;MASK $0F IF EVEN
F808:90 02      F80C            BCC  RTMASK
F80A:69 E0                      ADC  #$E0        ;MASK $F0 IF ODD
F80C:85 2E            RTMASK    STA  MASK
F80E:B1 26            PLOT1     LDA  (GBASL),Y   ;DATA
F810:45 30                      EOR  COLOR       ; XOR COLOR
F812:25 2E                      AND  MASK        ;  AND MASK
F814:51 26                      EOR  (GBASL),Y   ;   XOR DATA
F816:91 26                      STA  (GBASL),Y   ;    TO DATA
F818:60                         RTS
F819:                 *
F819:20 00 F8         HLINE     JSR  PLOT        ;PLOT SQUARE
F81C:C4 2C            HLINE1    CPY  H2          ;DONE?
F81E:B0 11      F831            BCS  RTS1        ; YES, RETURN
F820:C8                         INY              ; NO, INCR INDEX (X-COORD)
F821:20 0E F8                   JSR  PLOT1       ;PLOT NEXT SQUARE
F824:90 F6      F81C            BCC  HLINE1      ;ALWAYS TAKEN
F826:69 01            VLINEZ    ADC  #$01        ;NEXT Y-COORD
F828:48               VLINE     PHA              ; SAVE ON STACK
F829:20 00 F8                   JSR  PLOT        ; PLOT SQUARE
F82C:68                         PLA
F82D:C5 2D                      CMP  V2          ;DONE?
F82F:90 F5      F826            BCC  VLINEZ      ; NO, LOOP.
F831:60               RTS1      RTS
F832:                 *
F832:A0 2F            CLRSCR    LDY  #$2F        ;MAX Y, FULL SCRN CLR
F834:D0 02      F838            BNE  CLRSC2      ;ALWAYS TAKEN
F836:A0 27            CLRTOP    LDY  #$27        ;MAX Y, TOP SCRN CLR
F838:84 2D            CLRSC2    STY  V2          ;STORE AS BOTTOM COORD
F83A:                 ;                          FOR VLINE CALLS
F83A:A0 27                      LDY  #$27        ;RIGHTMOST X-COORD (COLUMN)
F83C:A9 00            CLRSC3    LDA  #$00        ;TOP COORD FOR VLINE CALLS
F83E:85 30                      STA  COLOR       ;CLEAR COLOR (BLACK)
F840:20 28 F8                   JSR  VLINE       ;DRAW VLINE
F843:88                         DEY              ;NEXT LEFTMOST X-COORD
F844:10 F6      F83C            BPL  CLRSC3      ;LOOP UNTIL DONE.
F846:60                         RTS
F847:                 *
F847:48               GBASCALC  PHA              ;FOR INPUT  00DEFGH
F848:4A                         LSR  A
F849:29 03                      AND  #$03
F84B:09 04                      ORA  #$04        ;GENERATE GBASH=000001FG
F84D:85 27                      STA  GBASH
F84F:68                         PLA              ;AND GBASL=HDEDE000
F850:29 18                      AND  #$18
F852:90 02      F856            BCC  GBCALC
F854:69 7F                      ADC  #$7F
F856:85 26            GBCALC    STA  GBASL
F858:0A                         ASL  A
F859:0A                         ASL  A
F85A:05 26                      ORA  GBASL
F85C:85 26                      STA  GBASL
F85E:60                         RTS
F85F:                 *
```

```
F85F:A5 30           NXTCOL   LDA   COLOR          ;INCREMENT COLOR BY 3
F861:18                       CLC
F862:69 03                    ADC   #$03
F864:29 0F           SETCOL   AND   #$0F           ;SETS COLOR=17*A MOD 16
F866:85 30                    STA   COLOR
F868:0A                       ASL   A              ;BOTH HALF BYTES OF COLOR EQUAL
F869:0A                       ASL   A
F86A:0A                       ASL   A
F86B:0A                       ASL   A
F86C:05 30                    ORA   COLOR
F86E:85 30                    STA   COLOR
F870:60                       RTS
F871:                *
F871:4A              SCRN     LSR   A              ;READ SCREEN Y-COORD/2
F872:08                       PHP                  ;SAVE LSB (CARRY)
F873:20 47 F8                 JSR   GBASCALC       ;CALC BASE ADDRESS
F876:B1 26                    LDA   (GBASL),Y      ;GET BYTE
F878:28                       PLP                  ;RESTORE LSB FROM CARRY
F879:90 04    F87F   SCRN2    BCC   RTMSKZ         ;IF EVEN, USE LO H
F87B:4A                       LSR   A
F87C:4A                       LSR   A
F87D:4A                       LSR   A              ;SHIFT HIGH HALF BYTE DOWN
F87E:4A                       LSR   A
F87F:29 0F           RTMSKZ   AND   #$0F           ;MASK 4-BITS
F881:60                       RTS
F882:                *
F882:A6 3A           INSDS1   LDX   PCL            ;PRINT PCL,H
F884:A4 3B                    LDY   PCH
F886:20 96 FD                 JSR   PRYX2
F889:20 48 F9                 JSR   PRBLNK         ;FOLLOWED BY A BLANK
F88C:A1 3A           INSDS2   LDA   (PCL,X)        ;GET OPCODE
F88E:A8                       TAY
F88F:4A                       LSR   A              ;EVEN/ODD TEST
F890:90 05    F897            BCC   IEVEN
F892:6A                       ROR   A              ;BIT 1 TEST
F893:B0 0C    F8A1            BCS   ERR            ;XXXXXX11 INVALID OP
F895:29 87                    AND   #$87           ;MASK BITS
F897:4A              IEVEN    LSR   A              ;LSB INTO CARRY FOR L/R TEST
F898:AA                       TAX
F899:BD 62 F9                 LDA   FMT1,X         ;GET FORMAT INDEX BYTE
F89C:20 79 F8                 JSR   SCRN2          ;R/L H-BYTE ON CARRY
F89F:D0 04    F8A5            BNE   GETFMT
F8A1:A0 FC           ERR      LDY   #$FC           ;SUBSTITUTE $FC FOR INVALID OPS
F8A3:A9 00                    LDA   #$00           ;SET PRINT FORMAT INDEX TO 0
F8A5:AA              GETFMT   TAX
F8A6:BD A6 F9                 LDA   FMT2,X         ;INDEX INTO PRINT FORMAT TABLE
F8A9:85 2E                    STA   FORMAT         ;SAVE FOR ADR FIELD FORMATTING
F8AB:29 03                    AND   #$03           ;MASK FOR 2-BIT LENGTH
F8AD:                ; (0=1 BYTE, 1=2 BYTE, 2=3 BYTE)
F8AD:85 2F                    STA   LENGTH
F8AF:20 35 FC                 JSR   NEWOPS       ;get index for new opcodes
F8B2:F0 18    F8CC            BEQ   GOTONE       ;found a new op (or no op)
F8B4:29 8F                    AND   #$8F           ;MASK FOR 1XXX1010 TEST
F8B6:AA                       TAX                  ; SAVE IT
F8B7:98                       TYA                  ;OPCODE TO A AGAIN
F8B8:A0 03                    LDY   #$03
F8BA:E0 8A                    CPX   #$8A
F8BC:F0 0B    F8C9            BEQ   MNNDX3
```

```
F8BE:4A             MNNDX1    LSR   A
F8BF:90 08    F8C9            BCC   MNNDX3          ;FORM INDEX INTO MNEMONIC TABLE
F8C1:4A                       LSR   A
F8C2:4A             MNNDX2    LSR   A               ;  1) 1XXX1010 => 00101XXX
F8C3:09 20                    ORA   #$20            ;  2) XXXYYY01 => 00111XXX
F8C5:88                       DEY                   ;  3) XXXYYY10 => 00110XXX
F8C6:D0 FA    F8C2            BNE   MNNDX2          ;  4) XXXYY100 => 00100XXX
F8C8:C8                       INY                   ;  5) XXXXX000 => 000XXXXX
F8C9:88             MNNDX3    DEY
F8CA:D0 F2    F8BE            BNE   MNNDX1
F8CC:60             GOTONE    RTS
F8CD:               *
F8CD:FF FF FF                 DFB   $FF,$FF,$FF
F8D0:               *
F8D0:20 82 F8       INSTDSP   JSR   INSDS1          ;GEN FMT, LEN BYTES
F8D3:48                       PHA                   ;SAVE MNEMONIC TABLE INDEX
F8D4:B1 3A          PRNTOP    LDA   (PCL),Y
F8D6:20 DA FD                 JSR   PRBYTE
F8D9:A2 01                    LDX   #$01            ;PRINT 2 BLANKS
F8DB:20 4A F9       PRNTBL    JSR   PRBL2
F8DE:C4 2F                    CPY   LENGTH          ;PRINT INST (1-3 BYTES)
F8E0:C8                       INY                   ;IN A 12 CHR FIELD
F8E1:90 F1    F8D4            BCC   PRNTOP
F8E3:A2 03                    LDX   #$03            ;CHAR COUNT FOR MNEMONIC INDEX
F8E5:C0 04                    CPY   #$04
F8E7:90 F2    F8DB            BCC   PRNTBL
F8E9:68                       PLA                   ;RECOVER MNEMONIC INDEX
F8EA:A8                       TAY
F8EB:B9 C0 F9                 LDA   MNEML,Y
F8EE:85 2C                    STA   LMNEM           ;FETCH 3-CHAR MNEMONIC
F8F0:B9 00 FA                 LDA   MNEMR,Y         ;  (PACKED INTO 2-BYTES)
F8F3:85 2D                    STA   RMNEM
F8F5:A9 00          PRMN1     LDA   #$00
F8F7:A0 05                    LDY   #$05
F8F9:06 2D          PRMN2     ASL   RMNEM           ;SHIFT 5 BITS OF CHARACTER INTO A
F8FB:26 2C                    ROL   LMNEM
F8FD:2A                       ROL   A               ;  (CLEARS CARRY)
F8FE:88                       DEY
F8FF:D0 F8    F8F9            BNE   PRMN2
F901:69 BF                    ADC   #$BF            ;ADD "?" OFFSET
F903:20 ED FD                 JSR   COUT            ;OUTPUT A CHAR OF MNEM
F906:CA                       DEX
F907:D0 EC    F8F5            BNE   PRMN1
F909:20 48 F9                 JSR   PRBLNK          ;OUTPUT 3 BLANKS
F90C:A4 2F                    LDY   LENGTH
F90E:A2 06                    LDX   #$06            ;CNT FOR 6 FORMAT BITS
F910:E0 03          PRADR1    CPX   #$03
F912:F0 1C    F930            BEQ   PRADR5          ;IF X=3 THEN ADDR.
F914:06 2E          PRADR2    ASL   FORMAT
F916:90 0E    F926            BCC   PRADR3
F918:BD B9 F9                 LDA   CHAR1-1,X
F91B:20 ED FD                 JSR   COUT
F91E:BD B3 F9                 LDA   CHAR2-1,X
F921:F0 03    F926            BEQ   PRADR3
F923:20 ED FD                 JSR   COUT
F926:CA             PRADR3    DEX
F927:D0 E7    F910            BNE   PRADR1
F929:60                       RTS
```

```
F92A:                   *
F92A:88                 PRADR4    DEY
F92B:30 E7     F914               BMI   PRADR2
F92D:20 DA FD                     JSR   PRBYTE
F930:A5 2E              PRADR5    LDA   FORMAT
F932:C9 E8                        CMP   #$E8             ;HANDLE REL ADR MODE
F934:B1 3A                        LDA   (PCL),Y          ;SPECIAL (PRINT TARGET,
F936:90 F2     F92A               BCC   PRADR4           ;  NOT OFFSET)
F938:20 56 F9           RELADR    JSR   PCADJ3
F93B:AA                           TAX                    ;PCL,PCH+OFFSET+1 TO A,Y
F93C:E8                           INX
F93D:D0 01     F940               BNE   PRNTYX           ;+1 TO Y,X
F93F:C8                           INY
F940:98                 PRNTYX    TYA
F941:20 DA FD           PRNTAX    JSR   PRBYTE           ;OUTPUT TARGET ADR
F944:8A                 PRNTX     TXA                    ;  OF BRANCH AND RETURN
F945:4C DA FD                     JMP   PRBYTE
F948:                   *
F948:A2 03              PRBLNK    LDX   #$03             ;BLANK COUNT
F94A:A9 A0              PRBL2     LDA   #$A0             ;LOAD A SPACE
F94C:20 ED FD           PRBL3     JSR   COUT             ;OUTPUT A BLANK
F94F:CA                           DEX
F950:D0 F8     F94A               BNE   PRBL2            ;LOOP UNTIL COUNT=0
F952:60                           RTS
F953:                   *
F953:38                 PCADJ     SEC                    ;0=1 BYTE, 1=2 BYTE,
F954:A5 2F              PCADJ2    LDA   LENGTH           ;  2=3 BYTE
F956:A4 3B              PCADJ3    LDY   PCH
F958:AA                           TAX                    ;TEST DISPLACEMENT SIGN
F959:10 01     F95C               BPL   PCADJ4           ;  (FOR REL BRANCH)
F95B:88                           DEY                    ;EXTEND NEG BY DECR PCH
F95C:65 3A              PCADJ4    ADC   PCL
F95E:90 01     F961               BCC   RTS2             ;PCL+LENGTH(OR DISPL)+1 TO A
F960:C8                           INY                    ;  CARRY INTO Y (PCH)
F961:60                 RTS2      RTS
F962:                   *
F962:                   ; FMT1 BYTES:     XXXXXXY0 INSTRS
F962:                   ; IF Y=0          THEN RIGHT HALF BYTE
F962:                   ; IF Y=1          THEN LEFT HALF BYTE
F962:                   ;                   (X=INDEX)
F962:                   *
F962:0F                 FMT1      DFB   $0F
F963:22                           DFB   $22
F964:FF                           DFB   $FF
F965:33                           DFB   $33
F966:CB                           DFB   $CB
F967:62                           DFB   $62
F968:FF                           DFB   $FF
F969:73                           DFB   $73
F96A:03                           DFB   $03
F96B:22                           DFB   $22
F96C:FF                           DFB   $FF
F96D:33                           DFB   $33
F96E:CB                           DFB   $CB
F96F:66                           DFB   $66
F970:FF                           DFB   $FF
F971:77                           DFB   $77
F972:0F                           DFB   $0F
```

```
F973:20                 DFB   $20
F974:FF                 DFB   $FF
F975:33                 DFB   $33
F976:CB                 DFB   $CB
F977:60                 DFB   $60
F978:FF                 DFB   $FF
F979:70                 DFB   $70
F97A:0F                 DFB   $0F
F97B:22                 DFB   $22
F97C:FF                 DFB   $FF
F97D:39                 DFB   $39
F97E:CB                 DFB   $CB
F97F:66                 DFB   $66
F980:FF                 DFB   $FF
F981:7D                 DFB   $7D
F982:0B                 DFB   $0B
F983:22                 DFB   $22
F984:FF                 DFB   $FF
F985:33                 DFB   $33
F986:CB                 DFB   $CB
F987:A6                 DFB   $A6
F988:FF                 DFB   $FF
F989:73                 DFB   $73
F98A:11                 DFB   $11
F98B:22                 DFB   $22
F98C:FF                 DFB   $FF
F98D:33                 DFB   $33
F98E:CB                 DFB   $CB
F98F:A6                 DFB   $A6
F990:FF                 DFB   $FF
F991:87                 DFB   $87
F992:01                 DFB   $01
F993:22                 DFB   $22
F994:FF                 DFB   $FF
F995:33                 DFB   $33
F996:CB                 DFB   $CB
F997:60                 DFB   $60
F998:FF                 DFB   $FF
F999:70                 DFB   $70
F99A:01                 DFB   $01
F99B:22                 DFB   $22
F99C:FF                 DFB   $FF
F99D:33                 DFB   $33
F99E:CB                 DFB   $CB
F99F:60                 DFB   $60
F9A0:FF                 DFB   $FF
F9A1:70                 DFB   $70
F9A2:24                 DFB   $24
F9A3:31                 DFB   $31
F9A4:65                 DFB   $65
F9A5:78                 DFB   $78
F9A6:         ; ZZXXXY01 INSTR'S
F9A6:00       FMT2      DFB   $00             ;ERR
F9A7:21                 DFB   $21             ;IMM
F9A8:81                 DFB   $81             ;Z-PAGE
F9A9:82                 DFB   $82             ;ABS
F9AA:59                 DFB   $59             ;(ZPAG,X)
F9AB:4D                 DFB   $4D             ;(ZPAG),Y
```

```
F9AC:91              DFB   $91       ;ZPAG,X
F9AD:92              DFB   $92        ;ABS,X
F9AE:86              DFB   $86        ;ABS,Y
F9AF:4A              DFB   $4A        ;(ABS)
F9B0:85              DFB   $85        ;ZPAG,Y
F9B1:9D              DFB   $9D        ;RELATIVE
F9B2:49              DFB   $49       ;(ZPAG)    (new)
F9B3:5A              DFB   $5A       ;(ABS,X)   (new)
F9B4:           *
F9B4:D9         CHAR2 DFB   $D9        ;'Y'
F9B5:00              DFB   $00       ; (byte F of FMT2)
F9B6:D8              DFB   $D8        ;'Y'
F9B7:A4              DFB   $A4        ;'$'
F9B8:A4              DFB   $A4        ;'$'
F9B9:00              DFB   $00
F9BA:           *
F9BA:AC         CHAR1 DFB   $AC        ;','
F9BB:A9              DFB   $A9        ;')'
F9BC:AC              DFB   $AC        ;','
F9BD:A3              DFB   $A3        ;'#'
F9BE:A8              DFB   $A8        ;'('
F9BF:A4              DFB   $A4        ;'$'
F9C0:1C         MNEML DFB   $1C
F9C1:8A              DFB   $8A
F9C2:1C              DFB   $1C
F9C3:23              DFB   $23
F9C4:5D              DFB   $5D
F9C5:8B              DFB   $8B
F9C6:1B              DFB   $1B
F9C7:A1              DFB   $A1
F9C8:9D              DFB   $9D
F9C9:8A              DFB   $8A
F9CA:1D              DFB   $1D
F9CB:23              DFB   $23
F9CC:9D              DFB   $9D
F9CD:8B              DFB   $8B
F9CE:1D              DFB   $1D
F9CF:A1              DFB   $A1
F9D0:1C              DFB   $1C       ;BRA
F9D1:29              DFB   $29
F9D2:19              DFB   $19
F9D3:AE              DFB   $AE
F9D4:69              DFB   $69
F9D5:A8              DFB   $A8
F9D6:19              DFB   $19
F9D7:23              DFB   $23
F9D8:24              DFB   $24
F9D9:53              DFB   $53
F9DA:1B              DFB   $1B
F9DB:23              DFB   $23
F9DC:24              DFB   $24
F9DD:53              DFB   $53
F9DE:19              DFB   $19
F9DF:A1              DFB   $A1       ; (A) FORMAT ABOVE
F9E0:AD              DFB   $AD       ; TSB
F9E1:1A              DFB   $1A
F9E2:5B              DFB   $5B
F9E3:5B              DFB   $5B
```

```
F9E4:A5                    DFB   $A5
F9E5:69                    DFB   $69
F9E6:24                    DFB   $24
F9E7:24                    DFB   $24          ; (B) FORMAT
F9E8:AE                    DFB   $AE
F9E9:AE                    DFB   $AE
F9EA:A8                    DFB   $A8
F9EB:AD                    DFB   $AD
F9EC:29                    DFB   $29
F9ED:8A                    DFB   $8A
F9EE:7C                    DFB   $7C
F9EF:8B                    DFB   $8B          ; (C) FORMAT
F9F0:15                    DFB   $15
F9F1:9C                    DFB   $9C
F9F2:6D                    DFB   $6D
F9F3:9C                    DFB   $9C
F9F4:A5                    DFB   $A5
F9F5:69                    DFB   $69
F9F6:29                    DFB   $29
F9F7:53                    DFB   $53          ; (D) FORMAT
F9F8:84                    DFB   $84
F9F9:13                    DFB   $13
F9FA:34                    DFB   $34
F9FB:11                    DFB   $11
F9FC:A5                    DFB   $A5
F9FD:69                    DFB   $69
F9FE:23                    DFB   $23           ; (E) FORMAT
F9FF:A0                    DFB   $A0
FA00:               *
FA00:D8             MNEMR  DFB   $D8
FA01:62                    DFB   $62
FA02:5A                    DFB   $5A
FA03:48                    DFB   $48
FA04:26                    DFB   $26
FA05:62                    DFB   $62
FA06:94                    DFB   $94
FA07:88                    DFB   $88
FA08:54                    DFB   $54
FA09:44                    DFB   $44
FA0A:C8                    DFB   $C8
FA0B:54                    DFB   $54
FA0C:68                    DFB   $68
FA0D:44                    DFB   $44
FA0E:E8                    DFB   $E8
FA0F:94                    DFB   $94
FA10:C4                    DFB   $C4          ;BRA
FA11:B4                    DFB   $B4
FA12:08                    DFB   $08
FA13:84                    DFB   $84
FA14:74                    DFB   $74
FA15:B4                    DFB   $B4
FA16:28                    DFB   $28
FA17:6E                    DFB   $6E
FA18:74                    DFB   $74
FA19:F4                    DFB   $F4
FA1A:CC                    DFB   $CC
FA1B:4A                    DFB   $4A
FA1C:72                    DFB   $72
```

```
FA1D:F2                         DFB   $F2
FA1E:A4                         DFB   $A4
FA1F:8A                         DFB   $8A            ; (A) FORMAT
FA20:06                         DFB   $06            ; TSB
FA21:AA                         DFB   $AA
FA22:A2                         DFB   $A2
FA23:A2                         DFB   $A2
FA24:74                         DFB   $74
FA25:74                         DFB   $74
FA26:74                         DFB   $74
FA27:72                         DFB   $72            ; (B) FORMAT
FA28:44                         DFB   $44
FA29:68                         DFB   $68
FA2A:B2                         DFB   $B2
FA2B:32                         DFB   $32
FA2C:B2                         DFB   $B2
FA2D:72                         DFB   $72
FA2E:22                         DFB   $22
FA2F:72                         DFB   $72            ; (C) FORMAT
FA30:1A                         DFB   $1A
FA31:1A                         DFB   $1A
FA32:26                         DFB   $26
FA33:26                         DFB   $26
FA34:72                         DFB   $72
FA35:72                         DFB   $72
FA36:88                         DFB   $88
FA37:C8                         DFB   $C8            ; (D) FORMAT
FA38:C4                         DFB   $C4
FA39:CA                         DFB   $CA
FA3A:26                         DFB   $26
FA3B:48                         DFB   $48
FA3C:44                         DFB   $44
FA3D:44                         DFB   $44
FA3E:A2                         DFB   $A2
FA3F:C8                         DFB   $C8            ; (E) FORMAT
FA40:                 *
FA40:48               IRQ       PHA                  ;save accumulator
FA41:68                         PLA                  ;rescued by stack trick later
FA42:68                         PLA
FA43:4C 06 C8                   JMP   IRQ1           ;do rest of IRQ handler
FA46:                 *
FA46:EA                         NOP
FA47:                 *
FA47:                 * NEWBRK is called by the interrupt handler which has
FA47:                 * set the hardware to its default state and encoded
FA47:                 * the state in the accumulator.  Software that wants
FA47:                 * to do break processing using full system resources
FA47:                 * can restore the machine state from this value.
FA47:                 *
FA47:85 44            NEWBRK    STA   MACSTAT        ;save state of machine
FA49:7A                         PLY                  ;restore registers for save
FA4A:FA                         PLX
FA4B:68                         PLA
FA4C:                 *
FA4C:28               BREAK     PLP                  ;Note: same as old BREAK routine!!
FA4D:20 4A FF                   JSR   SAVE           ;save reg's on BRK
FA50:68                         PLA                  ;including PC
FA51:85 3A                      STA   PCL
```

```
FA53:68                        PLA
FA54:85 3B                     STA   PCH
FA56:6C F0 03                  JMP   (BRKV)        ;call BRK HANDLER
FA59:               *
FA59:20 82 F8       OLDBRK     JSR   INSDS1         ;PRINT USER PC
FA5C:20 DA FA                  JSR   RGDSP1         ;  AND REGS
FA5F:4C 65 FF                  JMP   MON            ;GO TO MONITOR (NO PASS GO, NO $200!)
FA62:               *
FA62:D8             RESET      CLD                  ;DO THIS FIRST THIS TIME
FA63:20 84 FE                  JSR   SETNORM
FA66:20 2F FB                  JSR   INIT
FA69:20 93 FE                  JSR   SETVID
FA6C:20 89 FE                  JSR   SETKBD
FA6F:20 1C C4                  JSR   INITMOUSE     ;initialize the mouse
FA72:20 04 CC                  JSR   CLRPORT       ;clear port setup bytes
FA75:9C FF 04                  STZ   ACIABUF       ;and the commahead buffer
FA78:AD 5F C0                  LDA   SETAN3         ; AN3 = TTL HI
FA7B:20 BD FA                  JSR   RESET.X       ; initialize other devices
FA7E:2C 10 C0                  BIT   KBDSTRB        ; CLEAR KEYBOARD
FA81:D8             NEWMON     CLD
FA82:20 3A FF                  JSR   BELL           ; CAUSES DELAY IF KEY BOUNCES
FA85:AD F3 03                  LDA   SOFTEV+1       ;IS RESET HI
FA88:49 A5                     EOR   #$A5           ;A FUNNY COMPLEMENT OF THE
FA8A:CD F4 03                  CMP   PWREDUP        ; PWR UP BYTE ???
FA8D:D0 17     FAA6            BNE   PWRUP          ; NO SO PWRUP
FA8F:AD F2 03                  LDA   SOFTEV         ; YES SEE IF COLD START
FA92:D0 3B     FACF            BNE   NOFIX          ; HAS BEEN DONE YET?
FA94:A9 E0                     LDA   #$E0           ; DOES SEV POINT AT BASIC?
FA96:CD F3 03                  CMP   SOFTEV+1
FA99:D0 34     FACF            BNE   NOFIX          ; YES SO REENTER SYSTEM
FA9B:A0 03          FIXSEV     LDY   #3             ; NO SO POINT AT WARM START
FA9D:8C F2 03                  STY   SOFTEV         ; FOR NEXT RESET
FAA0:4C 00 E0                  JMP   BASIC          ; AND DO THE COLD START
FAA3:               *
FAA3:20 3A FF       BEEPFIX    JSR   BELL          ;Beep on powerup
FAA6:               *
FAA6:20 CA FC       PWRUP      JSR   COLDSTART     ;Trash memory, init ports
FAA9:          FAA9 SETPG3     EQU   *              ; SET PAGE 3 VECTORS
FAA9:A2 05                     LDX   #5
FAAB:BD FC FA       SETPLP     LDA   PWRCON-1,X     ; WITH CNTRL B ADRS
FAAE:9D EF 03                  STA   BRKV-1,X       ; OF CURRENT BASIC
FAB1:CA                        DEX
FAB2:D0 F7     FAAB            BNE   SETPLP
FAB4:A9 C6                     LDA   #$C6           ; LOAD HI SLOT +1
FAB6:80 5A     FB12            BRA   PWRUP2        ;branch around mnemonics
FAB8:               *
FAB8:               * Extension to MNEML (left mnemonics)
FAB8:               *
FAB8:8A                        DFB   $8A           ;PHY
FAB9:8B                        DFB   $8B           ;PLY
FABA:A5                        DFB   $A5           ;STZ
FABB:AC                        DFB   $AC           ;TRB
FABC:00                        DFB   $00           ;???
FABD:               *
FABD:               * This extension to the monitor reset routine ($FA62)
FABD:               * checks for apple keys.  If both are pressed, it goes
FABD:               * into an exerciser mode.  If the open apple key only is
FABD:               * pressed, memory is selectively trashed and a cold start
```

```
FABD:                * is done.
FABD:                *
FABD:A9 FF           RESET.X   LDA   #$FF
FABF:8D FB 04                  STA   VMODE         ;initialize mode
FAC2:0E 62 C0                  ASL   BUTN1
FAC5:2C 61 C0                  BIT   BUTN0
FAC8:10 64    FB2E             BPL   RTS2D
FACA:90 D7    FAA3             BCC   BEEPFIX       ;open apple only, reboot
FACC:4C 7C C7                  JMP   BANGER        ;both apples, exercise 'er
FACF:                *
FACF:6C F2 03        NOFIX     JMP   (SOFTEV)
FAD2:                *
FAD2:C1 D8 D9 D0     RTBL      ASC   'AXYPS'
FAD7:                *
FAD7:20 8E FD        REGDSP    JSR   CROUT          ;DISPLAY USER REG CONTENTS
FADA:A9 45           RGDSP1    LDA   #$45           ;WITH LABELS
FADC:85 40                     STA   A3L
FADE:A9 00                     LDA   #$00
FAE0:85 41                     STA   A3H
FAE2:A2 FB                     LDX   #$FB
FAE4:A9 A0           RDSP1     LDA   #$A0
FAE6:20 ED FD                  JSR   COUT
FAE9:BD D7 F9                  LDA   RTBL-251,X
FAEC:20 ED FD                  JSR   COUT
FAEF:A9 BD                     LDA   #$BD
FAF1:20 ED FD                  JSR   COUT
FAF4:B5 4A                     LDA   ACC+5,X
FAF6:80 0A    FB02             BRA   RGDSP2        ;make room for mnemonics
FAF8:                *
FAF8:                * Right half of new mnemonics, indexed from MNEMR
FAF8:                *
FAF8:74                        DFB   $74           ;PHY
FAF9:74                        DFB   $74           ;PLY
FAFA:76                        DFB   $76           ;STZ
FAFB:C6                        DFB   $C6           ;TRB
FAFC:00                        DFB   $00           ;???
FAFD:                *
FAFD:59 FA           PWRCON    DW    OLDBRK
FAFF:00 E0 45                  DFB   $00,$E0,$45
FB02:                *
FB02:20 DA FD        RGDSP2    JSR   PRBYTE
FB05:E8                        INX
FB06:30 DC    FAE4             BMI   RDSP1
FB08:60                        RTS
FB09:                *
FB09:C1 F0 F0 EC     TITLE     ASC   'Apple        ]['
FB11:C4                        DFB   $C4           ;optional filler
FB12:                *
FB12:86 00           PWRUP2    STX   LOC0           ; SETPG3 MUST RETURN X=0
FB14:85 01                     STA   LOC1           ; SET PTR H
FB16:20 60 FB                  JSR   APPLEII       ;Display our banner...
FB19:6C 00 00                  JMP   (LOC0)        ;JUMP $C600
FB1C:00                        BRK
FB1D:00                        BRK
FB1E:                *
FB1E:4C DE C7        PREAD     JMP   MPADDLE       ;read mouse paddle
FB21:A0 00                     LDY   #$00           ;INIT COUNT
FB23:EA                        NOP                  ;COMPENSATE FOR 1ST COUNT
```

```
FB24:EA              583          NOP
FB25:BD 64 C0        584 PREAD2   LDA   PADDLO,X       ;COUNT Y-REG EVERY 12 USEC.
FB28:10 04     FB2E  585          BPL   RTS2D
FB2A:C8              586          INY
FB2B:D0 F8     FB25  587          BNE   PREAD2         ;EXIT AT 255 MAX
FB2D:88              588          DEY
FB2E:60              589 RTS2D    RTS
FB2F:                 33          INCLUDE AUTOST2
```

```
FB2F:                   *
FB2F:A9 00              INIT      LDA   #$00          ;CLR STATUS FOR DEBUG SOFTWARE
FB31:85 48                        STA   STATUS
FB33:AD 56 C0                     LDA   LORES
FB36:AD 54 C0                     LDA   TXTPAGE1      ;INIT VIDEO MODE
FB39:AD 51 C0           SETTXT    LDA   TXTSET        ;SET FOR TEXT MODE
FB3C:A9 00                        LDA   #$00          ;FULL SCREEN WINDOW
FB3E:F0 0B     FB4B               BEQ   SETWND
FB40:AD 50 C0           SETGR     LDA   TXTCLR        ;SET FOR GRAPHICS MODE
FB43:AD 53 C0                     LDA   MIXSET        ;LOWER 4 LINES AS TEXT WINDOW
FB46:20 36 F8                     JSR   CLRTOP
FB49:A9 14                        LDA   #$14
FB4B:85 22              SETWND    STA   WNDTOP       ;SET WINDOW
FB4D:EA                           NOP
FB4E:EA                           NOP
FB4F:20 0A CE                     JSR   WNDREST      ;40/80 column width
FB52:80 05     FB59               BRA   VTAB23
FB54:                   *
FB54:09 80              DOCTL     ORA   #$80         ;controls need high bit
FB56:4C 54 CD                     JMP   CTLCHAR0     ;execute control char
FB59:                   *
FB59:A9 17              VTAB23    LDA   #$17          ;VTAB TO ROW 23
FB5B:85 25              TABV      STA   CV            ;VTABS TO ROW IN A-REG
FB5D:4C 22 FC                     JMP   VTAB         ;don't set OURCV!!
FB60:                   *
FB60:20 58 FC           APPLEII   JSR   HOME          ;CLEAR THE SCRN
FB63:A0 09                        LDY   #9
FB65:B9 02 FD           STITLE    LDA   APPLE2C-1,Y   ;GET A CHAR
FB68:99 0D 04                     STA   LINE1+13,Y    ;PUT IT AT TOP CENTER OF SCREEN
FB6B:88                           DEY
FB6C:D0 F7     FB65               BNE   STITLE
FB6E:60                           RTS
FB6F:                   *
FB6F:AD F3 03           SETPWRC   LDA   SOFTEV+1      ;ROUTINE TO CALCULATE THE 'FUNNY
FB72:49 A5                        EOR   #$A5          ;COMPLEMENT' FOR THE RESET VECTOR
FB74:8D F4 03                     STA   PWREDUP
FB77:60                           RTS
FB78:                   *
FB78:          FB78     VIDWAIT   EQU   *             ;CHECK FOR A PAUSE (CONTROL-S).
FB78:C9 8D                        CMP   #$8D          ;ONLY WHEN I HAVE A CR
FB7A:D0 18     FB94               BNE   NOWAIT        ;NOT SO, DO REGULAR
FB7C:AC 00 C0                     LDY   KBD           ;IS KEY PRESSED?
FB7F:10 13     FB94               BPL   NOWAIT        ;NO.
FB81:C0 93                        CPY   #$93          ;YES -- IS IT CTRL-S?
FB83:D0 0F     FB94               BNE   NOWAIT        ;NOPE - IGNORE
FB85:2C 10 C0                     BIT   KBDSTRB       ;CLEAR STROBE
FB88:AC 00 C0           KBDWAIT   LDY   KBD           ;WAIT TILL NEXT KEY TO RESUME
FB8B:10 FB     FB88               BPL   KBDWAIT       ;WAIT FOR KEYPRESS
FB8D:C0 83                        CPY   #$83          ;IS IT CONTROL-C?
FB8F:F0 03     FB94               BEQ   NOWAIT        ;YES, SO LEAVE IT
FB91:2C 10 C0                     BIT   KBDSTRB       ;CLR STROBE
FB94:2C 7B 06           NOWAIT    BIT   VFACTV       ;is video firmware active?
FB97:30 64     FBFD               BMI   VIDOUT       ;=>no, do normal 40 column
FB99:89 60                        BIT   #$60         ;is it a control?
FB9B:F0 B7     FB54               BEQ   DOCTL        ;=>yes, do it
FB9D:20 B8 C3                     JSR   STORCH       ;print w/inverse mask
FBA0:EE 7B 05           NEWADV    INC   OURCH        ;advance cursor
FBA3:AD 7B 05                     LDA   OURCH        ;and update others
```

```
FBA6:2C 1F C0                  BIT   RD80VID      ;but only if not 80 columns
FBA9:30 05     FBB0            BMI   NEWADV1      ;=>80 columns, leav'em
FBAB:8D 7B 04                  STA   OLDCH
FBAE:85 24                     STA   CH
FBB0:80 46     FBF8 NEWADV1    BRA   ADV2         ;check for CR
FBB2:               *
FBB2:EA                        NOP
FBB3:               *
FBB3:06             F8VERSION  DFB   GOODF8       ;//e, chels ID byte
FBB4:               *
FBB4:10 06     FBBC DOCOUT1    BPL   DCX          ;=>video firmware active, no mask
FBB6:C9 A0                     CMP   #$A0         ;is it control char?
FBB8:90 02     FBBC            BCC   DCX          ;=>yes, no mask
FBBA:25 32                     AND   INVFLG       ;else apply inverse mask
FBBC:4C F6 FD       DCX        JMP   COUTZ        ;and print character
FBBF:00                        BRK
FBC0:               *
FBC0:00                        DFB   $00          ;chels ID byte
FBC1:               *
FBC1:48             BASCALC    PHA                ;CALC BASE ADDR IN BASL,H
FBC2:4A                        LSR   A            ;FOR GIVEN LINE NO.
FBC3:29 03                     AND   #$03         ; 0<=LINE NO.<=$17
FBC5:09 04                     ORA   #$04         ;ARG=000ABCDE, GENERATE
FBC7:85 29                     STA   BASH         ; BASH=000001CD
FBC9:68                        PLA                ; AND
FBCA:29 18                     AND   #$18         ; BASL=EABAB000
FBCC:90 02     FBD0            BCC   BASCLC2
FBCE:69 7F                     ADC   #$7F
FBD0:85 28          BASCLC2    STA   BASL
FBD2:0A                        ASL   A
FBD3:0A                        ASL   A
FBD4:05 28                     ORA   BASL
FBD6:85 28                     STA   BASL
FBD8:60                        RTS
FBD9:               *
FBD9:C9 87          CHKBELL    CMP   #$87          ;BELL CHAR? (CONTROL-G)
FBDB:D0 12     FBEF            BNE   RTS2B         ; NO, RETURN.
FBDD:A9 40          BELL1      LDA   #$40          ; YES...
FBDF:20 A8 FC                  JSR   WAIT          ;DELAY .01 SECONDS
FBE2:A0 C0                     LDY   #$C0
FBE4:A9 0C          BELL2      LDA   #$0C          ;TOGGLE SPEAKER AT 1 KHZ
FBE6:20 A8 FC                  JSR   WAIT          ; FOR .1 SEC.
FBE9:AD 30 C0                  LDA   SPKR
FBEC:88                        DEY
FBED:D0 F5     FBE4            BNE   BELL2
FBEF:60             RTS2B      RTS
FBF0:               *
FBF0:A4 24          STORADV    LDY   CH           ;get 40 column position
FBF2:91 28                     STA   (BASL),Y     ;and store
FBF4:E6 24          ADVANCE    INC   CH           ;increment cursor
FBF6:A5 24                     LDA   CH
FBF8:C5 21          ADV2       CMP   WNDWDTH       ;BEYOND WINDOW WIDTH?
FBFA:B0 66     FC62            BCS   CR            ; YES, CR TO NEXT LINE.
FBFC:60             RTS3       RTS                 ; NO, RETURN.
FBFD:               *
FBFD:C9 A0          VIDOUT     CMP   #$A0          ;CONTROL CHAR?
FBFF:B0 EF     FBF0            BCS   STORADV       ; NO, OUTPUT IT.
FC01:A8                        TAY                 ;INVERSE VIDEO?
```

```
FC02:10 EC    FBF0              BPL   STORADV      ; YES, OUTPUT IT.
FC04:C9 8D          VIDOUT1     CMP   #$8D         ;CR?
FC06:F0 6B    FC73              BEQ   NEWCR        ;Yes, use new routine
FC08:C9 8A                      CMP   #$8A         ;LINE FEED?
FC0A:F0 5A    FC66              BEQ   LF           ; IF SO, DO IT.
FC0C:C9 88                      CMP   #$88         ;BACK SPACE? (CONTROL-H)
FC0E:D0 C9    FBD9              BNE   CHKBELL      ; NO, CHECK FOR BELL.
FC10:20 E2 FE       BS          JSR   DECCH        ;decrement all cursor H indices
FC13:10 E7    FBFC              BPL   RTS3         ;IF POSITIVE, OK; ELSE MOVE UP.
FC15:A5 21                      LDA   WNDWDTH      ;get window width,
FC17:20 EB FE                   JSR   WDTHCH       ;and set CH's to WNDWDTH-1
FC1A:A5 22          UP          LDA   WNDTOP       ;CURSOR V INDEX
FC1C:C5 25                      CMP   CV
FC1E:B0 DC    FBFC              BCS   RTS3         ;top line, exit
FC20:C6 25                      DEC   CV           ;not top, go up one
FC22:               *
FC22:80 62    FC86  VTAB        BRA   NEWVTAB      ;go update OURCV
FC24:20 C1 FB       VTABZ       JSR   BASCALC      ;calculate the base address
FC27:A5 20                      LDA   WNDLFT       ;get the left window edge
FC29:2C 1F C0                   BIT   RD80VID      ;80 columns?
FC2C:10 02    FC30              BPL   VTAB40       ;=>no, left edge ok
FC2E:4A                         LSR   A            ;divide width by 2
FC2F:18                         CLC                ;prepare to add
FC30:65 28          VTAB40      ADC   BASL         ;add width to base
FC32:85 28                      STA   BASL
FC34:60             RTS4        RTS
FC35:               *
FC35:               * NEWOPS translates the opcode in the Y register
FC35:               * to a mnemonic table index and returns with Z=1.
FC35:               * If Y is not a new opcode, Z=0.
FC35:               *
FC35:98             NEWOPS      TYA                ;get the opcode
FC36:A2 16                      LDX   #NUMOPS      ;check through new opcodes
FC38:DD FE FE       NEWOP1      CMP   OPTBL,X      ;does it match?
FC3B:F0 43    FC80              BEQ   GETINDX      ;=>yes, get new index
FC3D:CA                         DEX
FC3E:10 F8    FC38              BPL   NEWOP1       ;else check next one
FC40:60                         RTS                ;not found, exit with BNE
FC41:               *
FC41:00                         BRK
FC42:               *
FC42:80 19    FC5D  CLREOP      BRA   CLREOP1      ;ESC F IS CLR TO END OF PAGE
FC44:A5 25          CLREOP2     LDA   CV
FC46:48             CLEOP1      PHA                ;SAVE CURRENT LINE NO. ON STACK
FC47:20 24 FC                   JSR   VTABZ        ;CALC BASE ADDRESS
FC4A:20 9E FC                   JSR   CLEOLZ       ;CLEAR TO EOL. (SETS CARRY)
FC4D:A0 00                      LDY   #$00         ;CLEAR FROM H INDEX=0 FOR REST
FC4F:68                         PLA                ;INCREMENT CURRENT LINE NO.
FC50:1A                         INC   A
FC51:C5 23                      CMP   WNDBTM       ;DONE TO BOTTOM OF WINDOW?
FC53:90 F1    FC46              BCC   CLEOP1       ; NO, KEEP CLEARING LINES.
FC55:B0 CB    FC22              BCS   VTAB         ; YES, TAB TO CURRENT LINE
FC57:00                         BRK
FC58:               *
FC58:20 A5 CD       HOME        JSR   HOMECUR      ;move cursor home
FC5B:80 E7    FC44              BRA   CLREOP2      ;then clear to end of page
FC5D:               *
FC5D:20 9D CC       CLREOP1     JSR   GETCUR       ;load Y with proper CH
```

```
FC60:80 E2    FC44             BRA   CLREOP2      ;before clearing page
FC62:                    *
FC62:80 0F    FC73   CR        BRA   NEWCR        ;only LF if not Pascal
FC64:00                        BRK
FC65:00                        BRK
FC66:                    *
FC66:E6 25           LF        INC   CV            ;INCR CURSOR V. (DOWN 1 LINE)
FC68:A5 25                     LDA   CV
FC6A:C5 23                     CMP   WNDBTM        ;OFF SCREEN?
FC6C:90 1A    FC88             BCC   NEWVTABZ     ;set base+WNDLFT
FC6E:C6 25                     DEC   CV            ;DECR CURSOR V. (BACK TO BOTTOM)
FC70:                    *
FC70:4C 35 CB        SCROLL    JMP   SCROLLUP     ;scroll the screen
FC73:                    *
FC73:20 E9 FE        NEWCR     JSR   CLRCH        ;set CH's to 0
FC76:2C FB 04                  BIT   VMODE        ;is it Pascal?
FC79:10 0A    FC85             BPL   CRRTS        ;pascal, no LF
FC7B:20 44 FD                  JSR   NOESCAPE     ;else clear escape mode
FC7E:80 E6    FC66             BRA   LF           ;then do LF
FC80:                    *
FC80:BD 15 FF        GETINDX   LDA   INDX,X       ;lookup index for mnemonic
FC83:A0 00                     LDY   #0           ;exit with BEQ
FC85:60              CRRTS     RTS
FC86:                    *
FC86:A5 25           NEWVTAB   LDA   CV           ;update //e CV
FC88:8D FB 05        NEWVTABZ  STA   OURCV
FC8B:80 97    FC24             BRA   VTABZ        ;and calc base+WNDLFT
FC8D:                    *
FC8D:20 9D CC        NEWCLREOL JSR   GETCUR       ;get current cursor
FC90:A9 A0           NEWCLEOLZ LDA   #$A0         ;get a blank
FC92:2C 7B 06                  BIT   VFACTV       ;if video firmware active,
FC95:30 02    FC99             BMI   NEWC1        ;=>don't use inverse mask
FC97:25 32                     AND   INVFLG
FC99:4C C2 CB        NEWC1     JMP   DOCLR        ;go do clear
FC9C:                    *
FC9C:80 EF    FC8D   CLREOL    BRA   NEWCLREOL    ;get cursor and clear
FC9E:80 F0    FC90   CLEOLZ    BRA   NEWCLEOLZ    ;clear from Y
FCA0:                    *
FCA0:A0 00           CLRLIN    LDY   #0           ;clear entire line
FCA2:80 EC    FC90             BRA   NEWCLEOLZ
FCA4:                    *
FCA4:7C 2A CD        CTLDO     JMP   (CTLADR,X)   ;jump to proper routine
FCA7:                    *
FCA7:EA                        NOP
FCA8:                    *
FCA8:38              WAIT      SEC
FCA9:48              WAIT2     PHA
FCAA:E9 01           WAIT3     SBC   #$01
FCAC:D0 FC    FCAA             BNE   WAIT3          ;1.0204 USEC
FCAE:68                        PLA                  ;(13+2712*A+512*A*A)
FCAF:E9 01                     SBC   #$01
FCB1:D0 F6    FCA9             BNE   WAIT2
FCB3:60              RTS6      RTS
FCB4:                    *
FCB4:E6 42           NXTA4     INC   A4L            ;INCR 2-BYTE A4
FCB6:D0 02    FCBA             BNE   NXTA1          ; AND A1
FCB8:E6 43                     INC   A4H
FCBA:A5 3C           NXTA1     LDA   A1L            ;INCR 2-BYTE A1.
```

```
FCBC:C5 3E                     CMP   A2L            ; AND COMPARE TO A2
FCBE:A5 3D                     LDA   A1H            ; (CARRY SET IF >=)
FCC0:E5 3F                     SBC   A2H
FCC2:E6 3C                     INC   A1L
FCC4:D0 02    FCC8             BNE   RTS4B
FCC6:E6 3D                     INC   A1H
FCC8:60             RTS4B      RTS
FCC9:               *
FCC9:60             HEADR      RTS                  ;don't do it
FCCA:               *
FCCA:A0 B0          COLDSTART  LDY   #$B0           ;let it precess down
FCCC:64 3C                     STZ   A1L
FCCE:A2 BF                     LDX   #$BF           ;start from BFXX down
FCD0:86 3D          BLAST      STX   A1H
FCD2:A9 A0                     LDA   #$A0           ;store blanks
FCD4:91 3C                     STA   (A1L),Y
FCD6:88                        DEY
FCD7:91 3C                     STA   (A1L),Y
FCD9:CA                        DEX                  ;back down to next page
FCDA:E0 01                     CPX   #1             ;stay away from stack
FCDC:D0 F2    FCD0             BNE   BLAST          ;fall into COMINIT
FCDE:               *
FCDE:8D 01 C0                  STA   SET80COL       ;init ALT screen holes
FCE1:AD 55 C0                  LDA   TXTPAGE2       ;for serial and comm ports
FCE4:38                        SEC
FCE5:A2 88                     LDX   #$88
FCE7:BD 27 CB       COM1       LDA   COMTBL-1,X     ;XFER from rom
FCEA:90 0A    FCF6             BCC   COM2           ;branch if defaults ok
FCEC:DD 77 04                  CMP   $477,X         ;test for prior setup
FCEF:18                        CLC                  ;branch if not valid
FCF0:D0 04    FCF6             BNE   COM2           ;If $4F8 & $4FF = TBL values
FCF2:E0 82                     CPX   #$82
FCF4:90 06    FCFC             BCC   COM3
FCF6:9D 77 04       COM2       STA   $477,X
FCF9:CA                        DEX                  ;move all 8...
FCFA:D0 EB    FCE7             BNE   COM1
FCFC:AD 54 C0       COM3       LDA   TXTPAGE1       ;restore switches
FCFF:8D 00 C0                  STA   CLR80COL       ;to default states
FD02:60                        RTS
FD03:               *
FD03:                          MSB   ON
FD03:C1 F0 F0 EC    APPLE2C    ASC   "Apple         //c"
FD0C:               *
FD0C:A4 24          RDKEY      LDY   CH             ;get char at current position
FD0E:B1 28                     LDA   (BASL),Y       ;for those who restore it
FD10:EA                        NOP                  ;if a program controls input
FD11:EA                        NOP                  ;hooks, no cursor may be displayed
FD12:EA                        NOP
FD13:EA                        NOP
FD14:EA                        NOP
FD15:EA                        NOP
FD16:EA                        NOP
FD17:EA                        NOP
FD18:               *
FD18:6C 38 00       KEYIN0     JMP   (KSWL)         ;GO TO USER KEY-IN
FD1B:               *
FD1B:91 28          KEYIN      STA   (BASL),Y       ;erase false images
FD1D:20 4C CC                  JSR   SHOWCUR        ;display true cursor
```

```
FD20:20 70 CC        DONXTCUR  JSR  UPDATE       ;look for key, blink II cursor
FD23:10 FB    FD20             BPL  DONXTCUR     ;loop until keypress
FD25:48              GOTKEY    PHA               ;save character
FD26:A9 08                     LDA  #M.CTL       ;were escapes enabled?
FD28:2C FB 04                  BIT  VMODE
FD2B:D0 1D    FD4A             BNE  NOESC2       ;=>no, there is no escape
FD2D:68                        PLA               ;yes, there may be a way out!!
FD2E:C9 9B                     CMP  #ESC         ;escape?
FD30:D0 06    FD38             BNE  LOOKPICK     ;=>no escape
FD32:4C CC CC                  JMP  NEWESC       ;=>go do escape sequence
FD35:                *
FD35:4C ED CC        RDCHAR    JMP  ESCRDKEY     ;do RDKEY with escapes
FD38:                *
FD38:2C 7B 06        LOOKPICK  BIT  VFACTV       ;only process f.arrow
FD3B:30 07    FD44             BMI  NOESCAPE     ;if video firmware is active
FD3D:C9 95                     CMP  #PICK        ;was it PICK? (->,CTL-U)
FD3F:D0 03    FD44             BNE  NOESCAPE     ;no, just return
FD41:20 1D CC                  JSR  PICKY        ;yes, pick the character
FD44:                *
FD44:                * NOESCAPE is used by GETCOUT too.
FD44:                *
FD44:48              NOESCAPE  PHA               ;save it
FD45:A9 08           NOESC1    LDA  #M.CTL       ;disable escape sequences
FD47:0C FB 04                  TSB  VMODE        ;and enable controls
FD4A:68              NOESC2    PLA               ;by setting M.CTL
FD4B:60                        RTS
FD4C:                *
FD4C:EA                        NOP
FD4D:                *
FD4D:20 A6 C3        NOTCR     JSR  GETCOUT      ;disable controls and print
FD50:C9 88                     CMP  #$88          ;CHECK FOR EDIT KEYS
FD52:F0 1D    FD71             BEQ  BCKSPC        ;  - BACKSPACE
FD54:C9 98                     CMP  #$98
FD56:F0 0A    FD62             BEQ  CANCEL        ;  - CONTROL-X
FD58:E0 F8                     CPX  #$F8
FD5A:90 03    FD5F             BCC  NOTCR1        ;MARGIN?
FD5C:20 3A FF                  JSR  BELL          ; YES, SOUND BELL
FD5F:E8              NOTCR1    INX                ;ADVANCE INPUT INDEX
FD60:D0 13    FD75             BNE  NXTCHAR
FD62:A9 DC           CANCEL    LDA  #$DC          ;BACKSLASH AFTER CANCELLED LINE
FD64:20 A6 C3                  JSR  GETCOUT
FD67:20 8E FD        GETLNZ    JSR  CROUT         ;OUTPUT 'CR'
FD6A:A5 33           GETLN     LDA  PROMPT        ;OUTPUT PROMPT CHAR
FD6C:20 ED FD                  JSR  COUT
FD6F:A2 01           GETLN1    LDX  #$01          ;INIT INPUT INDEX
FD71:8A              BCKSPC    TXA
FD72:F0 F3    FD67             BEQ  GETLNZ        ;WILL BACKSPACE TO 0
FD74:CA                        DEX
FD75:20 ED CC        NXTCHAR   JSR  ESCRDKEY     ;do new RDCHAR (allow escapes)
FD78:C9 95                     CMP  #PICK         ;USE SCREEN CHAR
FD7A:D0 08    FD84             BNE  ADDINP        ; FOR CONTROL-U
FD7C:20 1D CC                  JSR  PICKY        ;lift char from screen
FD7F:EA                        NOP
FD80:EA                        NOP
FD81:EA                        NOP               ;no upshifting needed
FD82:EA                        NOP
FD83:EA                        NOP
FD84:9D 00 02        ADDINP    STA  IN,X          ;ADD TO INPUT BUFFER
```

```
FD87:C9 8D                      CMP   #$8D
FD89:D0 C2     FD4D             BNE   NOTCR
FD8B:20 9C FC        CROUT1     JSR   CLREOL         ;CLR TO EOL IF CR
FD8E:A9 8D           CROUT      LDA   #$8D
FD90:D0 5B     FDED             BNE   COUT           ;(ALWAYS)
FD92:                *
FD92:A4 3D           PRA1       LDY   A1H            ;PRINT CR,A1 IN HEX
FD94:A6 3C                      LDX   A1L
FD96:20 8E FD        PRYX2      JSR   CROUT
FD99:20 40 F9                   JSR   PRNTYX
FD9C:A0 00                      LDY   #$00
FD9E:A9 AD                      LDA   #$AD           ;PRINT '-'
FDA0:4C ED FD                   JMP   COUT
FDA3:                *
FDA3:A5 3C           XAM8       LDA   A1L
FDA5:09 07                      ORA   #$07           ;SET TO FINISH AT
FDA7:85 3E                      STA   A2L            ; MOD 8=7
FDA9:A5 3D                      LDA   A1H
FDAB:85 3F                      STA   A2H
FDAD:A5 3C           MOD8CHK    LDA   A1L
FDAF:29 07                      AND   #$07
FDB1:D0 03     FDB6             BNE   DATAOUT
FDB3:20 92 FD        XAM        JSR   PRA1
FDB6:A9 A0           DATAOUT    LDA   #$A0
FDB8:20 ED FD                   JSR   COUT           ;OUTPUT BLANK
FDBB:B1 3C                      LDA   (A1L),Y
FDBD:20 DA FD                   JSR   PRBYTE         ;OUTPUT BYTE IN HEX
FDC0:20 BA FC                   JSR   NXTA1
FDC3:90 E8     FDAD             BCC   MOD8CHK        ;NOT DONE YET. GO CHECK MOD 8
FDC5:60              RTS4C      RTS                  ;DONE.
FDC6:                *
FDC6:4A              XAMPM      LSR   A              ;DETERMINE IF MONITOR MODE IS
FDC7:90 EA     FDB3             BCC   XAM            ; EXAMINE, ADD OR SUBTRACT
FDC9:4A                         LSR   A
FDCA:4A                         LSR   A
FDCB:A5 3E                      LDA   A2L
FDCD:90 02     FDD1             BCC   ADD
FDCF:49 FF                      EOR   #$FF           ;FORM 2'S COMPLEMENT FOR SUBTRACT.
FDD1:65 3C           ADD        ADC   A1L
FDD3:48                         PHA
FDD4:A9 BD                      LDA   #$BD           ;PRINT '=', THEN RESULT
FDD6:20 ED FD                   JSR   COUT
FDD9:68                         PLA
FDDA:                *
FDDA:48              PRBYTE     PHA                  ;PRINT BYTE AS 2 HEX DIGITS
FDDB:4A                         LSR   A              ; (DESTROYS A-REG)
FDDC:4A                         LSR   A
FDDD:4A                         LSR   A
FDDE:4A                         LSR   A
FDDF:20 E5 FD                   JSR   PRHEXZ
FDE2:68                         PLA
FDE3:                *
FDE3:29 0F           PRHEX      AND   #$0F           ;PRINT HEX DIGIT IN A-REG
FDE5:09 B0           PRHEXZ     ORA   #$B0           ;LSBITS ONLY.
FDE7:C9 BA                      CMP   #$BA
FDE9:90 02     FDED             BCC   COUT
FDEB:69 06                      ADC   #$06
FDED:                *
```

```
FDED:6C 36 00        408 COUTT       JMP   (CSWL)         ;VECTOR TO USER OUTPUT ROUTINE
FDF0:                409 *
FDF0:2C 7B 06        410 COUT1       BIT   VFACTV         ;video firmware active?
FDF3:4C B4 FB        411             JMP   DOCOUT1        ;mask II mode characters
FDF6:84 35           412 COUTZ       STY   YSAV1          ;SAVE Y-REG
FDF8:48              413             PHA                  ;SAVE A -REG
FDF9:20 78 FB        414             JSR   VIDWAIT        ;OUTPUT CHR AND CHECK FOR CTRL-S
FDFC:68              415             PLA                  ;RESTORE A-REG
FDFD:A4 35           416             LDY   YSAV1          ;AND Y-REG
FDFF:60              417             RTS                  ;RETURN TO SENDER...
FE00:                418 *
FE00:C6 34           419 BL1         DEC   YSAV
FE02:F0 9F    FDA3   420             BEQ   XAM8
FE04:                421 *
FE04:CA              422 BLANK       DEX                   ;BLANK TO MON
FE05:D0 16    FE1D   423             BNE   SETMDZ          ;AFTER BLANK
FE07:C9 BA           424             CMP   #$BA            ;DATA STORE MODE?
FE09:D0 BB    FDC6   425             BNE   XAMPM           ; NO; XAM, ADD, OR SUBTRACT.
FE0B:                426 *
FE0B:85 31           427 STOR        STA   MODE            ;KEEP IN STORE MODE
FE0D:A5 3E           428             LDA   A2L
FE0F:91 40           429             STA   (A3L),Y         ;STORE AS LOW BYTE AT (A3)
FE11:E6 40           430             INC   A3L
FE13:D0 02    FE17   431             BNE   RTS5            ;INCR A3, RETURN.
FE15:E6 41           432             INC   A3H
FE17:60              433 RTS5        RTS
FE18:                434 *
FE18:A4 34           435 SETMODE     LDY   YSAV            ;SAVE CONVERTED ':', '+',
FE1A:B9 FF 01        436             LDA   IN-1,Y          ; '-', '.' AS MODE
FE1D:85 31           437 SETMDZ      STA   MODE
FE1F:60              438             RTS
FE20:                439 *
FE20:A2 01           440 LT          LDX   #$01
FE22:B5 3E           441 LT2         LDA   A2L,X           ;COPY A2 (2 BYTES) TO
FE24:95 42           442             STA   A4L,X           ; A4 AND A5
FE26:95 44           443             STA   A5L,X
FE28:CA              444             DEX
FE29:10 F7    FE22   445             BPL   LT2
FE2B:60              446             RTS
FE2C:                447 *
FE2C:B1 3C           448 MOVE        LDA   (A1L),Y         ;MOVE (A1) THRU (A2) TO (A4)
FE2E:91 42           449             STA   (A4L),Y
FE30:20 B4 FC        450             JSR   NXTA4
FE33:90 F7    FE2C   451             BCC   MOVE
FE35:60              452             RTS
FE36:                453 *
FE36:B1 3C           454 VERIFY      LDA   (A1L),Y         ;VERIFY (A1) THRU (A2)
FE38:D1 42           455             CMP   (A4L),Y         ; WITH (A4)
FE3A:F0 1C    FE58   456             BEQ   VFYOK
FE3C:20 92 FD        457             JSR   PRA1
FE3F:B1 3C           458             LDA   (A1L),Y
FE41:20 DA FD        459             JSR   PRBYTE
FE44:A9 A0           460             LDA   #$A0
FE46:20 ED FD        461             JSR   COUT
FE49:A9 A8           462             LDA   #$A8
FE4B:20 ED FD        463             JSR   COUT
FE4E:B1 42           464             LDA   (A4L),Y
FE50:20 DA FD        465             JSR   PRBYTE
```

```
FE53:A9 A9          466          LDA   #$A9
FE55:20 ED FD       467          JSR   COUT
FE58:20 B4 FC       468 VFYOK    JSR   NXTA4
FE5B:90 D9    FE36  469          BCC   VERIFY
FE5D:60             470          RTS
FE5E:               471 *
FE5E:20 75 FE       472 LIST     JSR   A1PC            ;MOVE A1 (2 BYTES) TO
FE61:A9 14          473          LDA   #$14            ; PC IF SPEC'D AND
FE63:48             474 LIST2    PHA                   ; DISASSEMBLE 20 INSTRUCTIONS.
FE64:20 D0 F8       475          JSR   INSTDSP
FE67:20 53 F9       476          JSR   PCADJ           ;ADJUST PC AFTER EACH INSTRUCTION.
FE6A:85 3A          477          STA   PCL
FE6C:84 3B          478          STY   PCH
FE6E:68             479          PLA
FE6F:38             480          SEC
FE70:E9 01          481          SBC   #$01            ;NEXT OF 20 INSTRUCTIONS
FE72:D0 EF    FE63  482          BNE   LIST2
FE74:60             483          RTS
FE75:               484 *
FE75:8A             485 A1PC     TXA                   ;IF USER SPECIFIED AN ADDRESS,
FE76:F0 07    FE7F  486          BEQ   A1PCRTS         ; COPY IT FROM A1 TO PC.
FE78:B5 3C          487 A1PCLP   LDA   A1L,X           ;YEP, SO COPY IT.
FE7A:95 3A          488          STA   PCL,X
FE7C:CA             489          DEX
FE7D:10 F9    FE78  490          BPL   A1PCLP
FE7F:60             491 A1PCRTS  RTS
FE80:               492 *
FE80:A0 3F          493 SETINV   LDY   #$3F            ;SET FOR INVERSE VID
FE82:D0 02    FE86  494          BNE   SETIFLG         ; VIA COUT1
FE84:A0 FF          495 SETNORM  LDY   #$FF            ;SET FOR NORMAL VID
FE86:84 32          496 SETIFLG  STY   INVFLG
FE88:60             497          RTS
FE89:               498 *
FE89:A9 00          499 SETKBD   LDA   #$00            ;DO 'IN#0'
FE8B:85 3E          500 INPORT   STA   A2L             ;DO 'IN#AREG'
FE8D:A2 38          501 INPRT    LDX   #KSWL
FE8F:A0 1B          502          LDY   #KEYIN
FE91:D0 08    FE9B  503          BNE   IOPRT
FE93:               504 *
FE93:A9 00          505 SETVID   LDA   #$0             ;DO 'PR#0'
FE95:85 3E          506 OUTPORT  STA   A2L             ;DO 'PR#AREG'
FE97:A2 36          507 OUTPRT   LDX   #CSWL
FE99:A0 F0          508          LDY   #COUT1
FE9B:A5 3E          509 IOPRT    LDA   A2L
FE9D:29 0F          510          AND   #$0F
FE9F:D0 06    FEA7  511          BNE   NOTPRT0        ;not slot 0
FEA1:C0 1B          512          CPY   #KEYIN         ;Continue if KEYIN
FEA3:F0 39    FEDE  513          BEQ   IOPRT1
FEA5:80 1B    FEC2  514          BRA   OPRT0          ;=>do PR#0
FEA7:09 C0          515 NOTPRT0  ORA   #<IOADR
FEA9:A0 00          516          LDY   #$00
FEAB:94 00          517 IOPRT2   STY   LOC0,X
FEAD:95 01          518          STA   LOC1,X
FEAF:60             519          RTS
FEB0:               520 *
FEB0:4C 00 E0       521 XBASIC   JMP   BASIC           ;TO BASIC, COLD START
FEB3:               522 *
FEB3:4C 03 E0       523 BASCONT  JMP   BASIC2          ;TO BASIC, WARM START
```

```
FEB6:              *
FEB6:20 75 FE      GO       JSR   A1PC          ;ADDR TO PC IF SPECIFIED
FEB9:20 3F FF               JSR   RESTORE       ;RESTORE FAKE REGISTERS
FEBC:6C 3A 00               JMP   (PCL)         ; AND GO!
FEBF:              *
FEBF:4C D7 FA      REGZ     JMP   REGDSP        ;GO DISPLAY REGISTERS
FEC2:              *
FEC2:3A            OPRT0    DEC   A             ;Need $FF
FEC3:8D FB 07               STA   CURSOR        ;set checkerboard cursor
FEC6:A9 F7                  LDA   #$FF-M.CTL    ;reset mode
FEC8:80 04    FECE          BRA   DOPRO
FECA:              *
FECA:4C F8 03      USR      JMP   USRADR        ;JUMP TO CONTROL-Y VECTOR IN RAM
FECD:              *
FECD:60            WRITE    RTS                 ;Tape write not needed
FECE:              *
FECE:8D 7B 06      DOPRO    STA   VFACTV        ;say video firmware inactive
FED1:8D 0E C0               STA   CLRALTCHAR    ;switch in normal char set
FED4:0C FB 04               TSB   VMODE         ;don't change M.CTL
FED7:DA                     PHX                 ;save X and Y
FED8:5A                     PHY                 ;for rest of PR#0
FED9:20 CD CD               JSR   CHK80         ;convert to 40 if needed
FEDC:7A                     PLY
FEDD:FA                     PLX
FEDE:A9 FD         IOPRT1   LDA   #<COUT1       ;set I/O page
FEE0:80 C9    FEAB          BRA   IOPRT2        ;=>go set output hook
FEE2:              *
FEE2:              * DECCH decrements the current cursor
FEE2:              * CLRCH sets all cursors to 0
FEE2:              * SETCUR sets cursors to value in Acc.
FEE2:              * See explanatory note with GETCUR
FEE2:              *
FEE2:5A            DECCH    PHY                 ;(from $FC10)
FEE3:20 9D CC               JSR   GETCUR        ;get current CH
FEE6:88                     DEY                 ;decrement it
FEE7:80 05    FEEE          BRA   SETCUR1       ;go update cursors
FEE9:              *
FEE9:A9 01         CLRCH    LDA   #1            ;set all cursors to 0
FEEB:3A            WDTHCH   DEC   A             ;dec window width (from $FC17)
FEEC:5A            SETCUR   PHY                 ;save Y
FEED:A8                     TAY                 ;need value in Y
FEEE:20 AD CC      SETCUR1  JSR   GETCUR2       ;save new CH
FEF1:7A                     PLY                 ;restore Y
FEF2:AD 7B 05               LDA   OURCH         ;and get new CH into acc
FEF5:60                     RTS                 ;(Need LDA to set flags)
FEF6:              *
FEF6:20 00 FE      CRMON    JSR   BL1           ;HANDLE CR AS BLANK
FEF9:68                     PLA                 ; THEN POP STACK
FEFA:68                     PLA                 ; AND RETURN TO MON
FEFB:D0 6C    FF69          BNE   MONZ          ;(ALWAYS)
FEFD:              *
FEFD:60            READ     RTS                 ;Tape read not needed
FEFE:              *
FEFE:              * OPTBL is a table containing the new opcodes that
FEFE:              * wouldn't fit into the existing lookup table.
FEFE:              *
FEFE:12            OPTBL    DFB   $12           ;ORA (ZPAG)
FEFF:14                     DFB   $14           ;TRB ZPAG
```

```
FF00:1A                         DFB   $1A          ;INC A
FF01:1C                         DFB   $1C          ;TRB ABS
FF02:32                         DFB   $32          ;AND (ZPAG)
FF03:34                         DFB   $34          ;BIT ZPAG,X
FF04:3A                         DFB   $3A          ;DEC A
FF05:3C                         DFB   $3C          ;BIT ABS,X
FF06:52                         DFB   $52          ;EOR (ZPAG)
FF07:5A                         DFB   $5A          ;PHY
FF08:64                         DFB   $64          ;STZ ZPAG
FF09:72                         DFB   $72          ;ADC (ZPAG)
FF0A:74                         DFB   $74          ;STZ ZPAG,X
FF0B:7A                         DFB   $7A          ;PLY
FF0C:7C                         DFB   $7C          ;JMP (ABS,X)
FF0D:89                         DFB   $89          ;BIT IMM
FF0E:92                         DFB   $92          ;STA (ZPAG)
FF0F:9C                         DFB   $9C          ;STZ ABS
FF10:9E                         DFB   $9E          ;STZ ABS,X
FF11:B2                         DFB   $B2          ;LDA (ZPAG)
FF12:D2                         DFB   $D2          ;CMP (ZPAG)
FF13:F2                         DFB   $F2          ;SBC (ZPAG)
FF14:FC                         DFB   $FC          ;??? (the unknown opcode)
FF15:           0016  NUMOPS    EQU   *-OPTBL-1    ;number of bytes to check
FF15:                 *
FF15:                 * INDX contains pointers to the mnemonics for each of
FF15:                 * the opcodes in OPTBL.  Pointers with BIT 7
FF15:                 * set indicate extensions to MNEML or MNEMR.
FF15:                 *
FF15:38               INDX      DFB   $38
FF16:FB                         DFB   $FB
FF17:37                         DFB   $37
FF18:FB                         DFB   $FB
FF19:39                         DFB   $39
FF1A:21                         DFB   $21
FF1B:36                         DFB   $36
FF1C:21                         DFB   $21
FF1D:3A                         DFB   $3A
FF1E:F8                         DFB   $F8
FF1F:FA                         DFB   $FA
FF20:3B                         DFB   $3B
FF21:FA                         DFB   $FA
FF22:F9                         DFB   $F9
FF23:22                         DFB   $22
FF24:21                         DFB   $21
FF25:3C                         DFB   $3C
FF26:FA                         DFB   $FA
FF27:FA                         DFB   $FA
FF28:3D                         DFB   $3D
FF29:3E                         DFB   $3E
FF2A:3F                         DFB   $3F
FF2B:FC                         DFB   $FC          ;???
FF2C:00                         BRK
FF2D:                 *
FF2D:A9 C5            PRERR     LDA   #$C5          ;PRINT 'ERR', THEN FALL INTO
FF2F:20 ED FD                   JSR   COUT          ; FWEEPER.
FF32:A9 D2                      LDA   #$D2
FF34:20 ED FD                   JSR   COUT
FF37:20 ED FD                   JSR   COUT
FF3A:                 *
```

```
FF3A:A9 87              BELL     LDA   #$87            ;MAKE A JOYFUL NOISE, THEN RETURN.
FF3C:4C ED FD                    JMP   COUT
FF3F:                   *
FF3F:A5 48              RESTORE  LDA   STATUS          ;RESTORE 6502 REGISTER CONTENTS
FF41:48                          PHA                   ; USED BY DEBUG SOFTWARE
FF42:A5 45                       LDA   A5H
FF44:A6 46              RESTR1   LDX   XREG
FF46:A4 47                       LDY   YREG
FF48:28                          PLP
FF49:60                          RTS
FF4A:                   *
FF4A:85 45              SAVE     STA   A5H             ;SAVE 6502 REGISTER CONTENTS
FF4C:86 46              SAV1     STX   XREG            ; FOR DEBUG SOFTWARE
FF4E:84 47                       STY   YREG
FF50:08                          PHP
FF51:68                          PLA
FF52:85 48                       STA   STATUS
FF54:BA                          TSX
FF55:86 49                       STX   SPNT
FF57:D8                          CLD
FF58:60                          RTS
FF59:                   *
FF59:20 84 FE           OLDRST   JSR   SETNORM         ;SET SCREEN MODE
FF5C:20 2F FB                    JSR   INIT            ; AND INIT KBD/SCREEN
FF5F:20 93 FE                    JSR   SETVID          ; AS I/O DEVS.
FF62:20 89 FE                    JSR   SETKBD
FF65:                   *
FF65:D8                 MON      CLD                   ;MUST SET HEX MODE!
FF66:20 3A FF                    JSR   BELL            ;FWEEPER.
FF69:A9 AA              MONZ     LDA   #$AA            ;'*' PROMPT FOR MONITOR
FF6B:85 33                       STA   PROMPT
FF6D:20 67 FD                    JSR   GETLNZ          ;READ A LINE OF INPUT
FF70:20 C7 FF                    JSR   ZMODE           ;CLEAR MONITOR MODE, SCAN IDX
FF73:20 A7 FF           NXTITM   JSR   GETNUM          ;GET ITEM, NON-HEX
FF76:84 34                       STY   YSAV            ; CHAR IN A-REG.
FF78:A0 13                       LDY   #SUBTBL-CHRTBL   ; X-REG=0 IF NO HEX INPUT
FF7A:88                 CHRSRCH  DEY
FF7B:30 E8    FF65               BMI   MON             ;COMMAND NOT FOUND, BEEP & TRY AGAIN.
FF7D:D9 CD FF                    CMP   CHRTBL,Y        ;FIND COMMAND CHAR IN TABLE
FF80:D0 F8    FF7A               BNE   CHRSRCH         ;NOT THIS TIME
FF82:20 BE FF                    JSR   TOSUB           ;GOT IT! CALL CORRESPONDING SUBROUTINE
FF85:A4 34                       LDY   YSAV            ;PROCESS NEXT ENTRY ON HIS LINE
FF87:4C 73 FF                    JMP   NXTITM
FF8A:                   *
FF8A:A2 03              DIG      LDX   #$03
FF8C:0A                          ASL   A
FF8D:0A                          ASL   A               ;GOT HEX DIGIT,
FF8E:0A                          ASL   A               ; SHIFT INTO A2
FF8F:0A                          ASL   A
FF90:0A                 NXTBIT   ASL   A
FF91:26 3E                       ROL   A2L
FF93:26 3F                       ROL   A2H
FF95:CA                          DEX                   ;LEAVE X=$FF IF DIG
FF96:10 F8    FF90               BPL   NXTBIT
FF98:A5 31              NXTBAS   LDA   MODE
FF9A:D0 06    FFA2               BNE   NXTBS2          ;IF MODE IS ZERO,
FF9C:B5 3F                       LDA   A2H,X           ; THEN COPY A2 TO A1 AND A3
FF9E:95 3D                       STA   A1H,X
```

```
FFA0:95 41                    STA   A3H,X
FFA2:E8             NXTBS2    INX
FFA3:F0 F3    FF98            BEQ   NXTBAS
FFA5:D0 06    FFAD            BNE   NXTCHR
FFA7:A2 00          GETNUM    LDX   #$00          ;CLEAR A2
FFA9:86 3E                    STX   A2L
FFAB:86 3F                    STX   A2H
FFAD:B9 00 02       NXTCHR    LDA   IN,Y          ;GET CHAR
FFB0:C8                       INY
FFB1:20 99 C3                 JSR   UPSHIFT0      ;upshift if necessary (set high bit)
FFB4:49 B0                    EOR   #$B0
FFB6:C9 0A                    CMP   #$0A
FFB8:90 D0    FF8A            BCC   DIG           ;it's a digit
FFBA:80 37    FFF3            BRA   GETHEX        ;check for other digits
FFBC:00                       BRK
FFBD:00                       BRK
FFBE:               *
FFBE:A9 FE          TOSUB     LDA   #<GO          ;DISPATCH TO SUBROUTINE, BY
FFC0:48                       PHA                 ; PUSHING THE HI-ORDER SUBR ADDR,
FFC1:B9 E0 FF                 LDA   SUBTBL,Y      ; THEN THE LO-ORDER SUBR ADDR
FFC4:48                       PHA                 ; ONTO THE STACK,
FFC5:A5 31                    LDA   MODE          ; (CLEARING THE MODE, SAVE THE OLD
FFC7:A0 00          ZMODE     LDY   #$00          ; MODE IN A-REG),
FFC9:84 31                    STY   MODE
FFCB:60                       RTS                 ; AND 'RTS' TO THE SUBROUTINE!
FFCC:               *
FFCC:EA                       NOP
FFCD:               *
FFCD:BC             CHRTBL    DFB   $BC           ;^C  (BASIC WARM START)
FFCE:B2                       DFB   $B2           ;^Y  (USER VECTOR)
FFCF:BE                       DFB   $BE           ;^E  (OPEN AND DISPLAY REGISTERS)
FFD0:EF                       DFB   $EF           ;V   (MEMORY VERIFY)
FFD1:C4                       DFB   $C4           ;^K  (IN#SLOT)
FFD2:A9                       DFB   $A9           ;^P  (PR#SLOT)
FFD3:BB                       DFB   $BB           ;^B  (BASIC COLD START)
FFD4:A6                       DFB   $A6           ;'-' (SUBTRACTION)
FFD5:A4                       DFB   $A4           ;'+' (ADDITION)
FFD6:06                       DFB   $06           ;M   (MEMORY MOVE)
FFD7:95                       DFB   $95           ;'<' (DELIMITER FOR MOVE, VFY)
FFD8:07                       DFB   $07           ;N   (SET NORMAL VIDEO)
FFD9:02                       DFB   $02           ;I   (SET INVERSE VIDEO)
FFDA:05                       DFB   $05           ;L   (DISASSEMBLE 20 INSTRS)
FFDB:00                       DFB   $00           ;G   (EXECUTE PROGRAM)
FFDC:93                       DFB   $93           ;':' (MEMORY FILL)
FFDD:A7                       DFB   $A7           ;'.' (ADDRESS DELIMITER)
FFDE:C6                       DFB   $C6           ;'CR' (END OF INPUT)
FFDF:99                       DFB   $99           ;BLANK
FFE0:               *
FFE0:               * Table of low order monitor routine
FFE0:               * dispatch addresses.
FFE0:               *
FFE0:B2             SUBTBL    DFB   >BASCONT-1
FFE1:C9                       DFB   >USR-1
FFE2:BE                       DFB   >REGZ-1
FFE3:35                       DFB   >VERIFY-1
FFE4:8C                       DFB   >INPRT-1
FFE5:96                       DFB   >OUTPRT-1
FFE6:AF                       DFB   >XBASIC-1
```

```
FFE7:17             756          DFB   >SETMODE-1
FFE8:17             757          DFB   >SETMODE-1
FFE9:2B             758          DFB   >MOVE-1
FFEA:1F             759          DFB   >LT-1
FFEB:83             760          DFB   >SETNORM-1
FFEC:7F             761          DFB   >SETINV-1
FFED:5D             762          DFB   >LIST-1
FFEE:B5             763          DFB   >GO-1
FFEF:17             764          DFB   >SETMODE-1
FFF0:17             765          DFB   >SETMODE-1
FFF1:F5             766          DFB   >CRMON-1
FFF2:03             767          DFB   >BLANK-1
FFF3:               768 *
FFF3:69 88          769 GETHEX   ADC   #$88
FFF5:C9 FA          770          CMP   #$FA
FFF7:B0 91    FF8A  771          BCS   DIG
FFF9:60             772          RTS
FFFA:               773 *
FFFA:FB 03          774          DW    NMI             ;NON-MASKABLE INTERRUPT VECTOR
FFFC:62 FA          775          DW    RESET           ;RESET VECTOR
FFFE:03 C8          776 IRQVECT  DW    NEWIRQ          ;INTERRUPT REQUEST VECTOR
```

```
   3D A1H             3C A1L            FE78 A1PCLP        FE7F A1PCRTS
 FE75 A1PC            3F A2H              3E A2L             41 A3H
   40 A3L             43 A4H              42 A4L             45 A5H
   44 A5L             45 ACC            C8FF ACDONE        04FF ACIABUF
 C988 ACIADONE      C900 ACIAINT        C908 AC'.ATST      FDD1 ADD
 FD84 ADDINP        FBF8 ADV2          ?FBF4 ADVANCE       C94B AIEATIT
 C943 AINOFLSH      C94D AIPASS         C922 AIPORT2       C91C AITST2
 C01E ALTCHARSET   ?03F5 AMPERV         FD03 APPLE2C       FB60 APPLEII
 0438 ASTAT         C6A2 BADRD1         C6D3 BADREAD       C77C BANGER
   2B BAS2H           2A BAS2L          FBC1 BASCALC       FBD0 BASCLC2
 FEB3 BASCONT         29 BASH           E003 BASIC2        C324 BASICENT
 C79F BASICIN       C317 BASICINIT      E000 BASIC           28 BASL
 FD71 BCKSPC        FAA3 BEEPFIX       ?FBDD BELL1         FF3A BELL
 FBE4 BELL2         0215 BINH           0214 BINL          C329 BINPUT
 FE00 BL1           FE04 BLANK          FCD0 BLAST           4F BOOTDEV
 07DB BOOTSCRN        3C BOOTTMP       ?C326 BPRINT       ?FA4C BREAK
 03F0 BRKV         ?FC10 BS               04 BUTMODE       C061 BUTN0
 C062 BUTN1         CFC2 C03            C307 C3COUT1      ?C300 C3ENTRY
 C305 C3KEYIN       FD62 CANCEL         CA76 CDONE2        CA3C CDONE
?CD7D CGO           F9BA CHAR1          F9B4 CHAR2         CDCD CHK80
 FBD9 CHKBELL       C528 CHKMOU         CB4E CHKRT         FF7A CHRSRCH
   24 CH            C132 CHOK           FFCD CHRTBL        CA28 CKDIG
 FC9E CLEOLZ        FC46 CLEOP1         CBEE CLR0          CBFC CLR1
 CBF1 CLR2          CC02 CLR3           CBC7 CLR40         C000 CLR80COL
 C00C CLR80VID      CBDA CLR80          C00E CLRALTCHAR   ?C058 CLRAN0
?C05A CLRAN1       ?C05C CLRAN2        ?C05E CLRAN3        FEE9 CLRCH
 C1DD CLRCOL        FC9C CLREOL         FC44 CLREOP2       FC42 CLREOP
 FC5D CLREOP1       CBCF CLRHALF        CD9B CLRIT         CC97 CLRKBD
 FCA0 CLRLIN        CC04 CLRPORT       ?CFFF CLRROM        F838 CLRSC2
 F83C CLRSC3       ?F832 CLRSCR         F836 CLRTOP        CA7D CMDB
 CA5D CMDCR           BF CMDCUR         CA79 CMDD          CA68 CMDI2
 CA67 CMDI          CA67 CMDK           CA14 CMDLOOP       CA67 CMDL
 C9DE CMDLIST       CA5D CMDN           CAB0 CMDP2         CA78 CMDP
 CAC4 CMDQ          CAB5 CMDR           CA99 CMDS          CAC6 CMDT
 CB05 CMDT2         CB17 CMDT3          C9C7 CMDTABLE      CA55 CMDZ
 CA25 CMDZ2         CA4D CMFOUND        C555 CMLOK         C538 CMLOOP
 C577 CMNOINT       C58E CMNOVBL        C57B CMNOY         C55D CMNTO
 C562 CMRGHT        C56F CMROK          CA43 CMSET         C542 CMXMOV
 CFB7 CO1           0738 COL            FCCA COLDSTART       30 COLOR
 FCE7 COM1          FCF6 COM2           FCFC COM3          CA36 COMINIT
 C9EB COMMAND       C266 COMMPORT       C263 COMOUT        C200 COMSLOT
 CB28 COMTBL        C338 COPYROM        C348 COPYROM2      FDF6 COUTZ
 FDED COUT          FDF0 COUT1          FEF6 CRMON        ?FD8B CROUT1
 FD8E CROUT         FC62 CR             FC85 CRRTS           37 CSWH
   36 CSWL          CD2A CTLADR         CD58 CTLCHAR       CD54 CTLCHAR0
 FCA4 CTLDO         CD6F CTLDONE        CD71 CTLGO         CD80 CTLGO1
   14 CTLNUM        CD91 CTLOFF         CD95 CTLON         CD15 CTLTAB
 07FB CURSOR        C51D CVBUT          C516 CVMOVED       C4ED CVNOVBL
   25 CV            FDB6 DATAOUT        FBBC DCX           FEE2 DECCH
 C2C8 DEFAULT       C2F1 DEFCOM         C2D9 DEFFF         C2FC DEFIDX
 C2CE DEFLOOP       C6D9 DENIB1         C6D7 DENIBL        C885 DEVNO
 FF8A DIG           CA30 DIGLOOP        0356 DNIBL         CBC2 DOCLR
 FBB4 DOCOUT1       FB54 DOCTL          C6FB DODRV2        C188 DONE
 FD20 DONXTCUR      FECE DOPRO         ?C701 DRV2BOOT      C60B DRV2ENT
 C5C3 DV10LOOP      C5CA DV10LT         C111 ENTR1         C230 ENTR
 F8A1 ERR             9B ESC            CCD7 ESC0         ?CCE3 ESC1
 CCE5 ESC2          CCC0 ESC3           CD0C ESCCHAR       0638 ESCHAR
 0013 ESCNUM        CCED ESCRDKEY       CCF8 ESCTAB        C28C EXIT1
 C28A EXITX        ?C65C EXTENT       C63D EXTENT1      05F9 EXTINT2
 0538 EXTINT        F800 F8ORG        FBB3 F8VERSION    C142 FIXCH
?FA9B FIXSEV        06B8 FLAGS        CB1B FLUSH        F962 FMT1
 F9A6 FMT2          CD67 FNDCTL         2E FORMAT       C64B FUG1
?C648 FUGIT         F847 GBASCALC       27 GBASH          26 GBASL
 F856 GBCALC        C8F8 GBEMPTY      C8DB GBNOOVR      C393 GETALT1
 C398 GETALT2       C37C GETALT       C8CC GETBUF       C3A6 GETCOUT
 GCA7 GETCUR1       CC9D GETCUR       CCAD GETCUR2      CCB7 GETCUR3
 CCBF GETCURX       F8A5 GETFMT       FFF3 GETHEX       FC80 GETINDX
?FD6F GETLN1        FD67 GETLNZ      ?FD6A GETLN        FFA7 GETNUM
 CB57 GETST         CEFA GETX        ?CF06 GETY         CF38 GKEY
?FD25 GOTKEY        FEB6 GO           C89F GOBREAK        06 GOODF8
 C28F GOREMOTE      C290 GOTERM       F8CC GOTONE         2C H2
 C64E HANGING       C5E3 HDDONE       C5BE HDLOOP       C5B8 HDPOS2
?FCC9 HEADR         C5AC HEXDEC2      C59B HEXTODEC    ?C057 HIRES
 F81C HLINE1       ?F819 HLINE        FC58 HOME         CDA5 HOMECUR
 CE1B HOOKITUP      CE20 HOOKUP       F897 IEVEN        0200 INBUF
 CA0C INCMD         FF15 INDX        ?F88C INSDS2       0200 IN
 C405 INENT         FB2F INIT         C41C INITMOUSE   ?FE8B INPORT
 FE8D INPRT         F882 INSDS1       F8D0 INSTDSP      CC12 INVERT
   32 INVFLG        CC1C INVX         C000 IOADR        FE9B IOPRT
 FEDE IOPRT1        FEAB IOPRT2       FF58 IORTS        C058 IOU
 C078 IOUDSBL       C079 IOUENBL      C806 IRQ1         C827 IRQ2
 C831 IRQ3          C83B IRQ4         C850 IRQ5         C85E IRQ6
 C861 IRQ7          C873 IRQ8         C88C IRQDNE1      C88F IRQDNE2
 C899 IRQDNE3       C882 IRQDONE     ?03FE IRQLOC       C989 IRQTBLE
 FFFE IRQVECT      ?FA40 IRQ          C663 ISMRK1       C22F ISRDY
 CFF9 JMPDEST       C32C JPINIT       C32F JPREAD       C335 JPSTAT
 C332 JPWRITE       C010 KBDSTRB      FB88 KBDWAIT      C000 KBD
 FD1B KEYIN        ?FD18 KEYIN0         39 KSWH           38 KSWL
 C08B LCBANK1       C083 LCBANK2        2F LENGTH       FC66 LF
 0400 LINE1         FE5E LIST         FE63 LIST2          2C LMNEM
   00 LOC0            01 LOC1         FD38 LOOKPICK     C056 LORES
 FE22 LT2           FE20 LT          ?  40 M.40           20 M.CTL2
   08 M.CTL           10 M.CURSOR       08 M.GOXY         01 M.MOUSE
   80 M.PASCAL        04 M.VMODE        44 MACSTAT      C709 MAKTBL
   2E MASK          C9D4 MASK1        C9D9 MASK2        05F8 MAXH
 04F8 MAXL          077D MAXXH        067D MAXXL       ?07FD MAXYH
?06FD MAXYL         C400 MBASIC       C79B MBBAD        0578 MINH
 0478 MINL          057D MINXH        047D MINXL       ?05FD MINYH
?04FD MINYL         C8AB MIRQLP       C8C2 MIRQSTD      C4F1 MISTAT
?C052 MIXCLR        C053 MIXSET       F9C0 MNEML        FA00 MNEMR
 F8BE MNNDX1        F8C2 MNNDX2       F8C9 MNNDX3       FDAD MOD8CHK
   31 MODE          FF65 MON          FF69 MONZ         067C MOUARM
 C063 MOUBUT        C048 MOUCLR      ?C058 MOUDSBL     ?C059 MOUENBL
 07FC MOUMODE       C4D5 MOUSEINT     CD9F MOUSOFF      CD99 MOUSON
 077C MOUSTAT       0478 MOUTEMP      C066 MOUX1        057C MOUXH
 C015 MOUXINT       047C MOUXL        C067 MOUY1        05FC MOUYH
 C017 MOUYINT       04FC MOUYL          20 MOVARM       CF86 MOVEAUX
 CF99 MOVEC2M       C8A2 MOVEIRQ      CF9F MOVELOOP     FE2C MOVE
 CFCB MOVERET       CF9F MOVESTRT       02 MOVMODE      C7DE MPADDLE
 C72F MSG           CAA6 MSLOOP       07F8 MSLOT        CAA4 MSWAIT
 0300 NBUF1         FBB0 NEWADV1      FBA0 NEWADV       FA47 NEWBRK
 FC99 NEWC1         FC90 NEWCLEOLZ    FC8D NEWCLREOL    FC73 NEWCR
 CCCC NEWESC        C803 NEWIRQ      ?FA81 NEWMON       FC38 NEWOP1
 FC35 NEWOPS        FC86 NEWVTAB      FC88 NEWVTABZ     CFA9 NEXTA1
 03FB NMI           CA09 NOCMD        C46B NOERROR     ?FD45 NOESC1
 FD4A NOESC2        FD44 NOESCAPE     C26B NOESC        FACF NOFIX
 C725 NOPATRN        C371 NOREAD         CA93 NOSHIFT        C4F9 NOSTAT2
 C36A NOT1           C8FE NOTACIA        FD5F NOTCR1         FD4D NOTCR
?CC68 NOTINV1        CC6B NOTINV2        CC53 NOTINV         FEA7 NOTPRTO
 C22E NOTRDY         FB94 NOWAIT         047F NUMBER         0016 NUMOPS
 FCBA NXTA1          FCB4 NXTA4          FF98 NXTBAS         FF90 NXTBIT
 FFA2 NXTBS2         FD75 NXTCHAR        FFAD NXTCHR        ?F85F NXTCOL
 077B NXTCUR         FF73 NXTITM         FA59 OLDBRK         047B OLDCH
 067A OLDCUR2        0679 OLDCUR        ?FF59 OLDRST         FEC2 OPRTO
 FEFE OPTBL          057B OURCH          05FB OURCV          C407 OUTENT
?FE95 OUTPORT        FE97 OUTPRT         C1E4 P1INIT         C1F3 P1READ2
 C1EE P1READ         C1FB P1STATUS       C1F6 P1WRITE        C211 P2INIT
 C213 P2READ         C217 P2STATUS       C215 P2WRITE        C064 PADDLO
 CF71 PASCALC       ?CF7F PASCLC2        CC0B PASINVERT      CF35 PASREAD
 C850 PASSKIP1       C97C PBFULL         C973 PBOK          ?F954 PCADJ2
 F95C PCADJ4         F953 PCADJ          F956 PCADJ3           3B PCH
   3A PCL            CF19 PCTL           C7F6 PDOK           C7EB PDON
 CC3D PICK1          CC33 PICK2          CC3F PICK3          CC4A PICK4
   95 PICK           CC1D PICKY          CF41 PINIT          CEBC PIORDY
 F800 PLOT           F80E PLOT1          CEC0 PNOTRDY        C402 PNULL
 FD92 PRA1           F910 PRADR1         F914 PRADR2         F926 PRADR3
 F92A PRADR4         F930 PRADR5         F94A PRBL2         ?F94C PRBL3
 F948 PRBLNK         FDDA PRBYTE        ?FB1E PREAD          FB25 PREAD2
?FF2D PRERR          CEF7 PRET          ?FDE3 PRHEX          FDE5 PRHEXZ
 F8F5 PRMN1          F8F9 PRMN2          C168 PRNOW         ?F941 PRNTAX
 F8DB PRNTBL         F8D4 PRNTOP         F940 PRNTYX         C14C PRNT
?F944 PRNTX            33 PROMPT         FD96 PRYX2          CF66 PS1
 CF51 PSETUP         CF54 PSETUP2        CF30 PSETX          C222 PSTAT2
 CEB1 PSTATUS        CEBE PSTERR        ?C070 PTRIG          C967 PUTBUF
 C7DA PUTINBUF       CE3B PVMODE         04B8 PWDTH          CEDD PWR1
 FAFD PWRCON         03F4 PWREDUP        CEF4 PWRET          CEC2 PWRITE
 CEF1 PWRITERET      FB12 PWRUP2         FAA6 PWRUP          C506 QLOOP
 C5E8 QTBL           CE45 QUIT           CE44 QX            ?C060 RD40SW
 C018 RD80COL        C01F RD80VID        C63F RDADR          C016 RDALTZP
 C6A8 RDAT0          C6AA RDAT1          C6BA RDAT2          C6BC RDAT3
 C6CB RDAT4          C6A6 RDATA          C003 RDCARDRAM     ?FD35 RDCHAR
 C642 RDDHDR         C656 RDHD0          C65E RDHD1          C667 RDHD2
 C671 RDHD3         ?C01D RDHIRES        FD0C RDKEY          C011 RDLCBNK2
 C012 RDLCRAM        C002 RDMAINRAM     ?C01B RDMIX          C01C RDPAGE2
 C013 RDRAMRD        C014 RDRAMWRT       C685 RDSEC1         C687 RDSEC2
 C68F RDSEC3         C683 RDSECT         FAE4 RDSP1          C01A RDTEXT
?C019 RDVBLBAR      ?FEFD READ           FAD7 REGDSP         FEBF REGZ
?F938 RELADR         FA62 RESET          FABD RESET.X        C354 RESETLC
 FF3F RESTORE       ?FF44 RESTR1         C641 RETRY1         C657 RETRY
 FADA RGDSP1         FB02 RGDSP2           2D RMNEM            4F RNDH
   4E RNDL           C081 ROMIN          C37B ROMOK          0478 ROMSTATE
 FAD2 RTBL           F80C RTMASK         F87F RTMSKZ         F831 RTS1
 FBEF RTS2B          F961 RTS2           FB2E RTS2D          FBFC RTS3
 FCC8 RTS4B         ?FDC5 RTS4C         ?FC34 RTS4           FE17 RTS5
?FCB3 RTS6          ?FF4C SAV1           FF4A SAVE           BFFB SCNTL
 BFFA SCOMD          CE58 SCR1           CE5E SCR2           CE66 SCR3
 CE79 SCR4           CE82 SCR5           CE8B SCR6           CE96 SCR7
?CE8D SCR8           CEAD SCR9           CBB9 SCRL3          CB9B SCRLEVEN
 CBA2 SCRLFT         CB6D SCRLIN         CBB0 SCRLODD       ?F871 SCRN
 F879 SCRN2          CE80 SCRN48         CE53 SCRN84         CB30 SCROLLDN
 CB38 SCROLLIT       CB35 SCROLLUP      ?FC70 SCROLL         BFF8 SDATA
 C61F SEEKZERO       C296 SERIN          C11E SERISOUT       03B8 SERMODE
 C18C SEROUT         C191 SEROUT2        C19D SEROUT3        C117 SERPORT
 C100 SERSLOT        C146 SERVID         CDC0 SET40          C001 SET80COL
 CDBE SET80             C00D SET80VID          C00F SETALTCHAR        C009 SETALTZP
?C059 SETAN0           ?C05B SETAN1           ?C05D SETAN2            C05F SETAN3
 C184 SETCH            ?F864 SETCOL            FEEE SETCUR1           FEEC SETCUR
 CB67 SETDBAS          ?FB40 SETGR             CE23 SETHOOKS          FE86 SETIFLG
 FE80 SETINV           ?C454 SETIOU            CDA1 SETIT             FE89 SETKBD
 FE1D SETMDZ            FE18 SETMODE           FE84 SETNORM          ?FAA9 SETPG3
 FAAB SETPLP           ?FB6F SETPWRC           C360 SETROM            CB88 SETSRC
 C008 SETSTDZP          CACD SETTERM          ?FB39 SETTXT            C233 SETUP
 CB83 SETUP2            FE93 SETVID            FB4B SETWND            CE1A SETX
 CBC1 SEV1              CC4C SHOWCUR           C45E SILOOP            C2BE SINOMOD
 C205 SIN               C465 SINOCH            CBA8 SKPLFT            CBB4 SKPRT
   2B SLOTZ               C1 SLTDMY            C46C SMINVALID         C18B SOCMD
 C1E2 SODONE            03F2 SOFTEV            C1BF SOMAIN            C1A9 SORDY
 C1C1 SORDY2            C207 SOUT              C030 SPKR                49 SPNT
 BFF9 SSTAT             CF29 STARTXY             48 STATUS            CAE6 STCLR
 FB65 STITLE           ?FE0B STOR              FBF0 STORADV           C3B8 STORCH
 C3DB STORE1           ?C3F7 STORE4            C3C1 STORE             C3EE STORE2
?C3F2 STORE3            C3F9 STORE5            C3B3 STORY             CB21 STRTS
 CAED STSET             CAF6 STWASOK           FFE0 SUBTBL            C246 SUDODEF
 C25C SUDONE            C249 SUNODEF           C257 SUOUT             C160 TAB
?FB5B TABV             ?C020 TAPEOUT           C740 TBL1              C749 TBL2
 C71B TBLLOOP2          C70D TBLLOOP           04F8 TEMP1             06F8 TEMP
 0578 TEMPA             05F8 TEMPY             C293 TERM1               DF TERMCUR
 C275 TESTKBD           0800 THBUF            ?FB09 TITLE             C15E TOOFAR
 FFBE TOSUB             06FF TRKEY             067F TRSER             05FF TWKEY
 057F TWSER             C050 TXTCLR            C054 TXTPAGE1          C055 TXTPAGE2
 C051 TXTSET            05FA TYPHED            CC91 UD2               C39B UPSHIFT
 FC1A UP                CC70 UPDATE            C399 UPSHIFT0          03F8 USRADR
 FECA USR                 2D V2                C070 VBLCLR            C019 VBLINT
   0C VBLMODE           FE36 VERIFY            067B VFACTV            FE58 VFYOK
 CE31 VIDMODE           FC04 VIDOUT1           FBFD VIDOUT            FB78 VIDWAIT
 F826 VLINEZ            F828 VLINE             04FB VMODE             FC30 VTAB40
 FC22 VTAB              FB59 VTAB23            FC24 VTABZ             FCA8 WAIT
 FCA9 WAIT2             FCAA WAIT3             FEEB WDTHCH            CDD5 WIN0
 CDE0 WIN1              CDED WIN2              CDF2 WIN3              CE02 WIN4
 CDD2 WIN40             CE18 WIN5              CDD4 WIN80               23 WNDBTM
   20 WNDLFT            CE0A WNDREST             22 WNDTOP              21 WNDWDTH
 C005 WRCARDRAM        ?FECD WRITE             C004 WRMAINRAM         CD8D X.CUR.OFF
 CD89 X.CUR.ON          CDB7 X.SI              CDB0 X.SO              C3A5 X.UPSHIFT
 FDB3 XAM               FDA3 XAM8              FDC6 XAMPM             FEB0 XBASIC
 C9AD XBITKBD           C9C0 XBKB1             C9C2 XBKB2             06FB XCOORD
 CFF6 XFERAZP           CFE0 XFERC2M           CFE6 XFERZP            CFCD XFER
 C752 XLOOP1            C780 XMBASIC           C78D XMBOUT            C490 XMCDONE
 C4B0 XMCLAMP           C484 XMCLEAR           C59A XMDONE            C473 XMH2
 C471 XMHLOOP           C46D XMHOME            C4CF XMINT             C763 XMODE
 C495 XMREAD            C50C XMSKIP            C4FC XMTSTINT          C9A8 XNOKEY
 C8F0 XNOSBUF             91 XON               C758 XPAGE             C4A4 XRBUT
 C4AB XRBUT2            C98F XRDKBD            C8C5 XRDSER            C8FF XRDSNO
   46 XREG              C9A0 XRKBD1            C423 XRLOOP            C766 XRSET
 C76E XRST1             C43D XSETMOU           C452 XSOFF            ?C100 XXX
 0008 YHI                 47 YREG                35 YSAV1               34 YSAV
 FFC7 ZMODE             C7FB ZZNM1             CB24 ZZNM2             CE4D ZZQUIT
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED  4406
** FREE SPACE PAGE COUNT   47
```

Glossary

65C02: The microprocessor used in the Apple IIc computer.

ACIA: Asynchronous Communications Interface Adapter. A single chip that converts data from parallel to serial form, and vice versa, and handles serial transmission and reception and RS-232-C signals, under the control of its internal registers set and changed by firmware or software.

accumulator: The register in the 6502 and 65C02 microprocessors where most computations are performed.

acronym: A word formed from the initial letters of a name or phrase, such as ROM, from read-only memory.

ADC: See **analog-to-digital converter**.

address: A number used to identify something, such as a location in the computer's memory.

analog: Represented in terms of a physical quantity that can vary smoothly and continuously over a range of values. For example, a conventional 12-hour clock face is an analog device that represents the time of day in terms of the angles of the clock's hands. Compare **digital**.

analog-to-digital converter: A device that converts quantities from analog to digital form. For example, the Apple IIc's hand control converts the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes in steps even when the dial is turned smoothly.

AND: A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare **OR, exclusive OR, NOT**.

Apple IIc: A personal computer in the Apple II family, manufactured and sold by Apple Computer, Inc.

Applesoft: An extended version of the BASIC programming language used with the Apple IIc computer. The firmware for interpreting and executing programs in Applesoft is included in the Apple IIc ROM.

ASCII: American Standard Code for Information Interchange; a code in which the numbers from 0 to 127 stand for text characters, used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language.

assembly language: A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

asserted: Made true (positive in positive-true logic; negative in negative-true logic).

asynchronous: Having a variable time interval between characters.

back panel: The rear face of the Apple IIc computer, which includes the power switch, the power connector, and connectors for two serial devices, a video display device, an external disk drive, and a mouse or hand control.

bandwidth: A measure of the range of frequencies a device can handle. In the case of a video monitor, greater bandwidth enables it to display more information; to display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

base address: In indexed addressing, the fixed component of an address.

baud: A unit of signaling speed equal to the number of discrete conditions or signal events per second. Often equated (though not precisely) with bits per second.

binary: The representation of numbers in terms of powers of two, using the two digits 0 and 1. Commonly used in computers, since the values 0 and 1 can easily be represented in physical form in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen.

bit: A binary digit (0 or 1); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

board: See **printed-circuit board**.

boot: To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to *pull itself up by its own bootstraps*.

bootstrap: See **boot**.

BREAK: A SPACE (0) signal sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a timesharing service.

BRK: A 65C02 instruction that causes the microprocessor to halt.

buffer: An area of the computer's memory used as a *holding area* where information can be stored by one program or device and then read out by another at a different speed.

bus: A group of wires that transmit related information from one part of a computer system to another. In the Apple IIc, the address bus has 16 wires, and the data bus has eight.

byte: A unit of information consisting of a fixed number of bits; on the Apple IIc, one byte consists of eight bits and can represent any value between 0 and 255.

carriage return: An ASCII character (decimal 13; Appendix H) that ordinarily causes a printer or display device to place the subsequent character on the left margin. On a manual typewriter, this movement is combined with line feed (the advancement of the paper to the next line). With computers, carriage return and line feed are separate, causing hair-raising problems for the user.

carrier: The background signal on a communication channel that is modified to *carry* the information. Under RS-232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

carry flag: The C bit in the 65C02 processor status register, used to hold the *carry bit* in addition and subtraction.

cathode-ray tube: An electronic device, such as a television picture tube, that produces images on a screen coated with phosphors that emit light when struck by a focused beam of electrons.

central processing unit: See **processor**.

character: A letter, digit, punctuation mark, or other symbol used in printing, displaying or transferring information.

character code: A number used to represent a text character for processing by a computer system.

chip: The small piece of semiconducting material (usually silicon) on which an integrated circuit is fabricated.

Clear To Send: An RS-232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out.

code: (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

cold start: The process of starting up the Apple IIc when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, then loading and running a program. Compare **warm start**.

command: A communication from the user to a computer system (usually typed from the keyboard) directing it to perform some action.

command character: An ASCII character, usually CONTROL-A or CONTROL-I, that causes the serial port firmware to interpret subsequent characters as a command.

command register An ACIA location (at address $C09A for port 1 and $C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

communication mode: An operating state in which serial port 2 (or 1, if so set) is prepared to exchange data and signals with a DCE (such as a modem).

compiler: A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) for later execution. Compare **interpreter**.

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it.

computer: An electronic device for performing predefined (programmed) computations at high speed and with great accuracy.

computer system: A computer and its associated hardware, firmware, and software.

connector: A physical device such as a plug, socket, or jack, used to connect two devices to one another.

control character: A character that controls or modifies the way information is printed or displayed. Control characters have ASCII codes between $00 and $1F (or between $80 and $9F if the high-order bit is set). You can generate them at the Apple IIc keyboard by holding down CONTROL while typing one of the letter keys or @ [\ ] ^ or _.

control register: An ACIA location (at address $C09B for port 1, or $C0AB for port 2) that stores data format and baud rate selections.

CPU: Central processing unit; see **processor**.

CRT: See **cathode-ray tube**.

cursor: A symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital-to-analog converter**.

data: Information; especially information used or operated on by a program.

data bit: One of five to eight bits representing a character.

Data Carrier Detect: An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIc) indicating that a communication connection has been established.

Data Communication Equipment: As defined by the RS-232-C standard, any device that transmits or receives information. Usually this is a modem. However, when a Modem Eliminator is used, the Apple IIc itself looks like a DCE to the other device, and the other device looks like a DCE to the Apple IIc.

data format: The form in which data is stored, manipulated or transferred. Serial data transmitted and received by port 1 or 2 has a data format of: one start bit, five to eight data bits, an optional parity bit, and one, one and a half, or two stop bits.

Data Set Ready: An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection.

Data Terminal Equipment: As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as a terminus of a communication connection.

Data Terminal Ready: An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data.

DCD: See **Data Carrier Detect**.

DCE: See **Data Communication Equipment**.

debug: To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems.

decimal: The common form of number representation used in everyday life, in which numbers are expressed in terms of powers of ten, using the ten digits 0 to 9.

default: A value, action, or setting that is assumed or set in the absence of explicit instructions otherwise.

demodulate: To recover the information being transmitted by a modulated signal; for example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into sound emitted by a speaker.

device: (1) A physical apparatus for performing a particular task or achieving a particular purpose. (2) In particular, a hardware component of a computer system.

digit: (1) One of the characters 0 to 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 to 9 and A to F in hexadecimal.

digital: Represented in a discrete (noncontinuous) form, such as numerical digits. For example, contemporary digital clocks display the time in numerical form (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog.**

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIP: See **dual in-line package**.

disassembler: A language translator that converts a machine-language program into an equivalent program in assembly language, more easily understood by a human programmer. The opposite of an assembler.

disk: An information storage medium consisting of a flat, circular magnetic surface on which information can be recorded in the form of small magnetized spots, similarly to the way sounds are recorded on tape.

disk drive: A device that writes and reads information on the surface of a magnetic disk.

diskette: A term sometimes used for the small (5-1/4-inch) flexible disks used with the Apple Disk II drive.

Disk II drive: A model of disk drive made and sold by Apple Computer for use with the Apple IIe computer; uses 5-1/4-inch flexible (*floppy*) disks.

Disk Operating System: An optional software system for the Apple IIe that enables the computer to control and communicate with one or more Disk II drives.

display: (1) Information exhibited visually, especially on the screen of a display device. (2) To exhibit information visually. (3) A display device.

display device: A device that exhibits information visually, such as a television receiver or video monitor.

display screen: The glass or plastic panel on the front of a display device, on which images are displayed.

DOS: See **Disk Operating System**.

DSR: See **Data Set Ready**.

DTE: See **Data Terminal Equipment**.

DTR: See **Data Terminal Ready**.

dual in-line package: An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side; similar in appearance to an armored centipede.

echo: To send an input character to a video display, printer, or other output device.

edit: To change or modify; for example, to insert, remove, replace, or move text in a document.

editor: A program that enables the user to create and edit information of a particular form; for example, a *text editor* or a *graphics editor*.

effective address: In machine-language programming, the address of the memory location on which a particular instruction actually operates, which may be arrived at by indexed addressing or some other addressing method.

emulation mode: A manner of operating in which one computer or interface imitates another.

even parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits (among the data bits plus the parity bit) an even number.

error message: A message displayed or printed to notify the user of an error or problem in the execution of a program.

escape mode: A state of the Apple IIe computer, entered by pressing the (ESC) key, in which certain keys on the keyboard take on special meanings for positioning the cursor and controlling the display of text on the screen.

escape sequence: A sequence of keystrokes, beginning with (ESC), used for positioning the cursor and controlling the display of text on the screen.

exclusive OR: A logical operator that produces a true result if one of its operands is true and the other false, a false result if its operands are both true or both false; compare **OR, AND, NOT**.

execute: To perform or carry out a specified action or sequence of actions, such as those described by a program.

firmware: Software stored permanently in hardware: programs in read-only memory (ROM). Such programs (for example, the Applesoft interpreter and the Apple IIc Monitor program) are built into the computer at the factory; they can be executed at any time but cannot be modified or erased from main memory. Compare **hardware, software**.

fixed-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number, so that the number is interpreted as an integer. Compare **floating-point**.

flexible disk: A disk made of flexible plastic; often called a *floppy* disk. Compare **rigid disk**.

floating-point: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to *float* to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point**.

form feed: An ASCII character (decimal 12; Appendix H) that causes a printer or other paper-handling device to advance to the top of the next page.

framing error: In serial data transfer, absence of the expected stop bit(s) at the end of a received character. The serial port 1 and 2 ACIAs record this error by setting bit 1 (FRM) of its status register to 1. The ACIA checks and records each framing error separately: if the next character is OK, the FRM bit is cleared.

full duplex: Capable of simultaneous two-way communication.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

half duplex: Capable of communication in one direction at a time.

hand control: An optional peripheral device that can be connected to the Apple IIc's hand control connector and has a rotating dial and a pushbutton; typically used to control game-playing programs, but can be used in more serious applications as well.

hand control connector: A 9-pin connector on the Apple IIc's back panel, used for connecting hand controls to the computer.

hardware: Those components of a computer system consisting of physical (electronic or mechanical) devices. Compare **software, firmware**.

hertz: The unit of frequency of vibration or oscillation, also called cycles per second; named for the physicist Heinrich Hertz and abbreviated Hz. The Apple IIc's 65C02 microprocessor operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

hexadecimal: The representation of numbers in terms of powers of sixteen, using the sixteen digits 0 to 9 and A to F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form: each hexadecimal digit corresponds to a sequence of four binary digits, or bits.

high-level language: A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language.

high-order byte: The more significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

high-resolution graphics: The display of graphics on the Apple IIc's display screen as a six-color array of points, 280 columns wide and 192 rows high.

hold time: In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off; compare **setup time**.

Hz: See **hertz**.

IC: See **integrated circuit**.

index: (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

indexed addressing: A method of specifying memory addresses used in machine-language programming.

index register: A register in a computer processor that holds an index for use in indexed addressing. The Apple IIc's 65C02 microprocessor has two index registers, called the X register and the Y register.

input: (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integer: A whole number, with no fractional part; represented inside the computer in fixed-point form.

integrated circuit: An electronic component consisting of many circuit elements fabricated on a single piece of semiconducting material, such as silicon; see **chip**.

interface: The devices, rules, or conventions by which one component of a system communicates with another.

interpreter: A language translator that reads a program written in a particular programming language and immediately carries out the actions that the program describes. Compare **compiler**.

interrupt: A temporary suspension in the execution of a program by a computer in order to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

inverse video: The display of text on the computer's display screen in the form of black dots on a white (or other single phosphor color) background, instead of the usual white dots on a black background.

I/O: Input/output; the transfer of information into and out of a computer. See **input, output**.

I/O device: Input/output device; a device that transfers information into or out of a computer. See **input, output, peripheral device**.

I/O link: A fixed location that contains the address of an input/output subroutine in the Apple IIc Monitor program.

K: Two to the tenth power, or 1024 (from the Greek root *kilo*, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

keyboard: The set of keys built into the Apple IIc computer, similar to a typewriter keyboard, for typing information to the computer.

keystroke: The act of pressing a single key or a combination of keys (such as CONTROL-C) on the Apple IIc keyboard.

kilobyte: A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see **K**.

KSW: The symbolic name of the location in the Apple IIc's memory where the standard input link is stored; stands for *keyboard switch*. See **I/O link**.

language: See **programming language**.

language translator: A system program that reads a program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See **interpreter, compiler, assembler**.

least significant bit: The right-hand bit of a binary number as written down; it's positional value is 0 or 1.

line feed: An ASCII character (decimal 10; Appendix H) that ordinarily causes a printer or video display to advance to the next line.

load: To transfer information from a peripheral storage medium (such as a disk) into main memory for use; for example, to transfer a program into memory for execution.

local: Nearby; capable of direct connection using wires only.

location: See **memory location**.

logical operator: An operator, such as **AND**, that combines logical values to produce a logical result.

low-level language: A programming language that is relatively close to the form that the computer's processor can execute directly. Low-level languages available for the Apple IIc include 65C02 machine language and 65C02 assembly language.

low-order byte: The less significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

low-power Schottky: A type of TTL integrated circuit having lower power and higher speed than a conventional TTL integrated circuit.

low-resolution graphics: The display of graphics on the Apple IIc's display screen as a sixteen-color array of blocks, 40 columns wide and 48 rows high.

machine language: The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 65C02 microprocessor used in the Apple IIc) has its own form of machine language.

main memory: The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device can then check for errors by looking for this value on each character.

memory: A hardware component of a computer system that can store information for later retrieval; see **main memory, random-access memory, read-only memory, read-write memory**.

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size; in the Apple IIc, a memory location holds one byte, or eight bits, of information.

MHz: Megahertz; one million hertz. See **hertz**.

microcomputer: A computer, such as the Apple IIc, whose processor is a microprocessor.

microprocessor: A computer processor contained in a single integrated circuit, such as the 65C02 microprocessor used in the Apple IIc.

microsecond: One millionth of a second; abbreviated us.

millisecond: One thousandth of a second; abbreviated ms.

mode: A state of a computer or system that determines its behavior.

modem: Modulator/demodulator; a peripheral device that enables the computer to transmit and receive information over a telephone line; a DCE that connects a DTE to communication lines.

modem eliminator: The physical crossing of wires that replaces a pair of modems for direct connection of two DTEs.

modulate: To modify or alter a signal so as to transmit information; for example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or AM) or the frequency (frequency modulation, or FM) of a carrier signal.

monitor: See **video monitor**.

Monitor program: A system program built into the Apple IIc in firmware, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

most significant bit: The leftmost bit of a binary number as written down. This bit represents 0 or 1 times 2 to the power one less than the total number of bits in the binary number. For example, in the binary number 10000, which contains five digits, the *1* represents 1 times two to the fourth power—or sixteen.

nanosecond: One billionth (in British usage, one thousand-millionth) of a second; abbreviated ns.

network: A collection of interconnected, individually controlled computers, together with the hardware and software used to connect them.

nibble: A unit of information equal to half a byte, or four bits; can hold any value from 0 to 15. Sometimes spelled *nybble*.

NOT: A unary logical operator that produces a true result if its operand is false, a false result if its operand is true; compare **AND, OR, exclusive OR**.

NTSC: (1) National Television Standards Committee; the committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

object code: See **object program**.

object program: The translated form of a program produced by a language translator such as a compiler or assembler; also called object code. Compare **source program**.

odd parity: Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number.

opcode: See **operation code**.

operand: A value to which an operator is applied; the value on which an opcode operates.

operating system: A software system that organizes the computer's resources and capabilities and makes them available to the user or to application programs running on the computer.

operation code: The part of a machine-language instruction that specifies the operation to be performed; often called **opcode**.

operator: A symbol or sequence of characters, such as + or *AND*, specifying an operation to be performed on one or more values (the operands) to produce a result.

OR: A logical operator that produces a true result if either or both of its operands are true, a false result if both of its operands are false; compare **exclusive OR, AND, NOT**.

output: Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

overrun: A condition that occurs when the Apple IIc processor does not retrieve a received character from the ACIAs receive data register before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

page: (1) A screenful of information on a video display, consisting on the Apple IIc of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256 bytes.

page zero: See **zero page**.

parallel interface: An interface in which many bits of information (typically eight bits, or one byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

parity: Maintenance of a sameness of level or count, usually the count of 1 bits in each character, for error checking.

parity error: Absence of the correct parity bit value in a received character. The serial port ACIAs record this error by setting bit 0 (PAR) of their status registers to 1.

PC board: See **printed-circuit board**.

phase: (1) A stage in a periodic process; a point in a cycle; for example, the 65C02 microprocessor uses a clock cycle consisting of two phases called PHI0 and PHI1. (2) The relationship between two periodic signals or processes; for example, in NTSC color video, the color of a point on the screen is expressed by the instantaneous phase of the video signal relative to the color reference signal.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All other things equal, processors that have this feature run faster than those without it.

pointer: An item of information consisting of the memory address of some other item.

pop: To remove the top entry from a stack.

port: The point of connection, usually a physical connector, between a computer and a peripheral device, another computer, or a network.

power supply: The hardware component of a computer that draws electrical power from a power outlet and converts it to the forms needed by some other hardware component.

printed-circuit board: A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, from which all conducting material except the desired circuits is etched, and to which integrated circuits and other electronic components are connected.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

program: (1) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (2) To write a program.

programming language: A set of rules or conventions for writing programs.

prompt: To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.

prompt character: A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character **]** is used by the Applesoft BASIC interpreter, > by Integer BASIC, and * by the system Monitor program.

prompt message: A message displayed on the screen to prompt the user for some action.

protocol: A predefined exchange of control sighals between devices enabling them to prepare for and carry out coordinated data transfers.

push: To add an entry to the top of a stack.

radio-frequency modulator: A device for converting the video signals produced by a computer to a form that can be accepted by a television receiver.

RAM: See **random-access memory**.

random-access memory: Memory in which the contents of individual locations can be referred to in an arbitrary or random order.

raster: The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive dots on the individual lines of the raster.

read: To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

read-only memory: Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and can never be erased or changed. Compare **read-write memory, random-access memory, write-only memory**.

read-write memory: Memory whose contents can be both read and written; often misleadingly called *random-access memory*, or *RAM*. The information contained in read-write memory is erased when the computer's power is turned off, and is permanently lost unless it has been saved on a more permanent storage medium, such as a disk. Compare **read-only memory, random-access memory, write-only memory**.

receive data register: A read-only register in each serial port ACIA (at location $C098 for port 1 and $C0A8 for port 2) that stores the most recent character successfully received.

register: A location in a computer processor where an item of information, such as a byte, is held and modified under program control. Registers in the 65C02 microprocessor include the accumulator (A), two index registers (X and Y), the stack pointer (S), the processor status register (P), and the program counter (PC). The PC register holds two bytes (sixteen bits); the other registers hold one byte (eight bits) each.

remote: Too distant for direct connection using wires or cables only.

Request To Send: An RS-232-C signal from a DTE to a DCE to prepare the DCE for data transmission.

return address: The point in a program to which control returns on completion of a subroutine.

RF modulator: See **radio-frequency modulator**.

RI: See **Ring Indicator**.

rigid disk: A disk made of a hard, nonflexible material. Compare **flexible disk**.

Ring Indicator: An optional RS-232-C signal from a DCE to a DTE that indicates the arrival of a call.

ROM: See **read-only memory**.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

RS-232-C: A standard created by the Electronic Industries Association (EIA) to allow devices of different manufacturers to exchange serial data—particularly via telephone lines.

RTS: See **Request To Send**.

run: (1) To execute a program. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To transfer information from main memory to a peripheral storage medium for later use.

screen: See **display screen**.

scroll: To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See **viewport, window**.

serial interface: An interface in which information is transmitted sequentially, one bit at a time, over a single wire or channel. Compare **parallel interface**.

setup time: The amount of time a signal must be valid in advance of some event; compare **hold time**.

silicon: A non-metallic, semiconducting chemical element from which integrated circuits are made.

soft switch: A means of changing some feature of the Apple IIc from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

software: Those components of a computer system consisting of programs that determine or control the behavior of the computer. Compare **hardware, firmware**.

source code: See **source program**.

source program: The original form of a program given to a language translator such as a compiler or assembler for conversion into another form; sometimes called *source code*. Compare **object program**.

space character: A text character whose printed representation is a blank space, typed from the keyboard by pressing the SPACE bar.

SPACE parity: A bit of value 0 appended to a binary number for transmission. The receiving device can look for this value on each character as a means of error checking.

stack: A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in-first-out) order.

start bit: A transition from a MARK signal to a SPACE signal for one bit-time, indicating that the next string of bits represents a character.

status register: A register in an ACIA (at location $C099 for port 1 and $C0A9 for port 2) that stores the state of two of the RS-232-C signals and the state of the transmit and receive data registers, as well as the outcome of the most recent character transfer.

stop bit: A MARK signal following a string of data bits (or their optional parity bit) to indicate the end of a character.

string: An item of information consisting of a sequence of text characters.

strobe: (1) An event, such as a change in a signal, that triggers some action. (2) A signal whose change is used to trigger some action.

subroutine: A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

television receiver: A display device capable of receiving broadcast video signals (such as commercial television) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple IIc computer. Compare **video monitor**.

television set: See **television receiver**.

terminal: A device consisting of a typewriterlike keyboard and a display device, used for communicating between a computer system and a human user. Personal computers such as the Apple IIc typically have all or part of a terminal built into them.

terminal mode: An operating state of the Apple IIc communication port in which the firmware makes the computer act like a simple ASCII terminal.

text: (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple IIc's display screen. Compare **graphics**.

text window: An area on the Apple IIc's display screen within which text is displayed and scrolled.

transistor-to-transistor logic: (1) A family of integrated circuits used in computers and related devices. (2) A standard for interconnecting such circuits that defines the voltages used to represent logical zeros and ones.

transmit data register: A write-only register in one of the serial port ACIAs (at location $C098 for port 1 and $C0A8 for port 2) that holds the current character to be transmitted.

troubleshoot: To locate and correct the cause of a problem or malfunction in a computer system. Typically used to refer to hardware-related problems; compare **debug**.

TTL: See **transistor-to-transistor logic**.

unary operator: An operator that applies to a single operand; for example, the minus sign (–) in a negative number such as –6 is a unary arithmetic operator.

user: The person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.

vector: (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the Apple IIc computer as a display device. Compare **television receiver**.

viewport: All or part of the display screen, used by an application program to display a portion of the information (such as a document, picture, or worksheet) that the program is working on. Compare **window**.

warm start: The process of restarting the Apple IIc after the power is already on, without reloading the operating system into main memory and often without losing the program or information already in main memory. Compare **cold start**.

window: The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen; compare **viewport**.

word: A group of bits of a fixed size that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

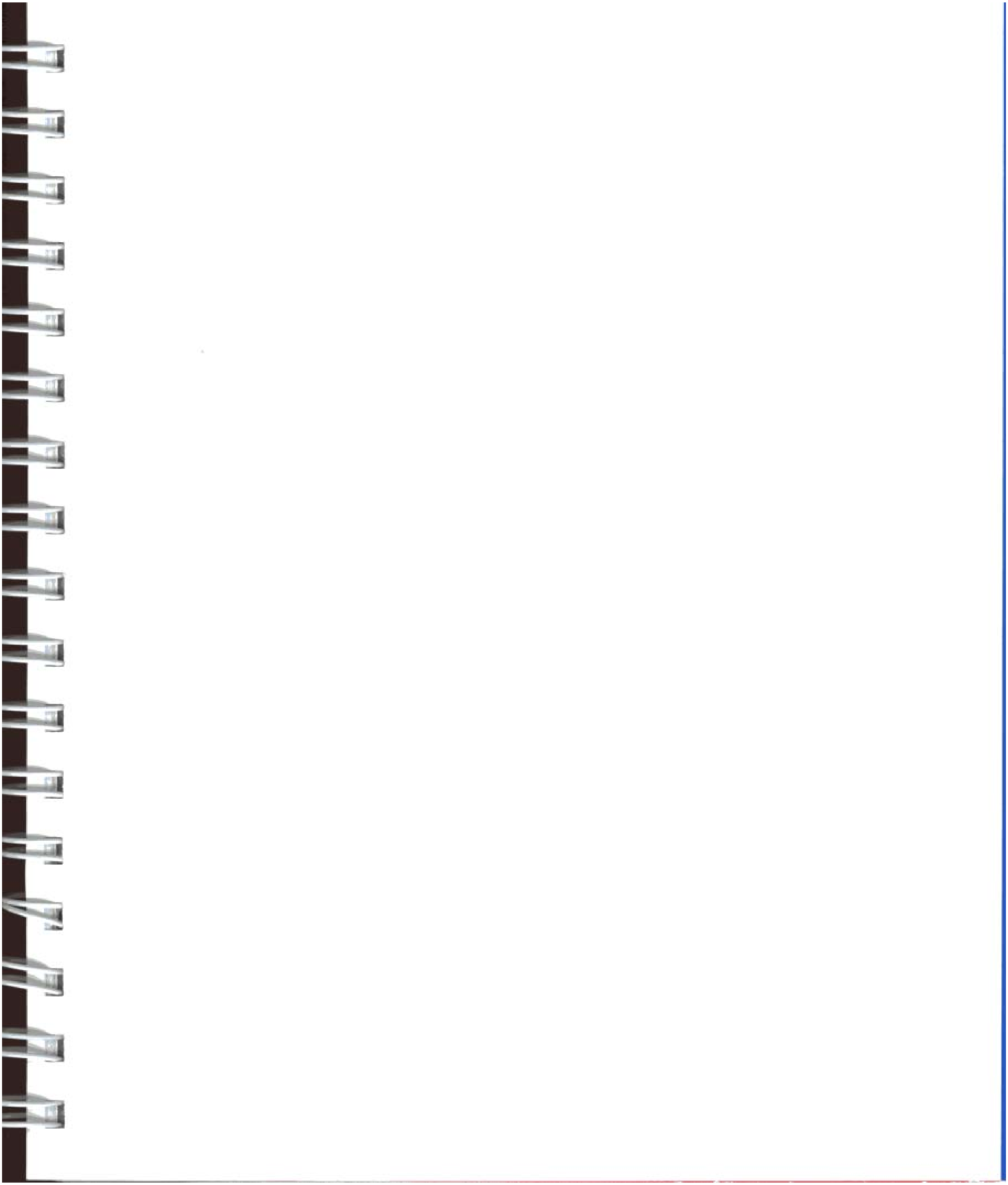
wraparound: The automatic continuation of text from the end of one line to the beginning of the next, as on the display screen or a printer.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

X register: One of the index registers in the 65C02 microprocessor.

Y register: One of the index registers in the 65C02 microprocessor.

zero page: The first page (256 bytes) of the Apple IIc's memory, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.



Bibliography

Apple II Monitors Peeled. Cupertino, Calif.: Apple Computer, Inc., 1978.

Currently not updated for Apple IIe and IIc, but a good introduction to Apple II series input/output procedures; also useful for historical background.

Apple IIe Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1982.

Addendum to the Design Guidelines. Cupertino, Calif.: Apple Computer, Inc., 1984.

Apple IIe Reference Manual. Cupertino, Calif.: Apple Computer, Inc., 1982.

Applesoft BASIC Programmer's Reference Manual, Volumes 1 and 2. For the Apple II, IIe, and IIc. Cupertino, Calif.: Apple Computer, Inc., 1982.

The version that applies to both the Apple IIe and the Apple IIc has Apple product number A2L0084 (Vol. 1) and A2L0085 (Vol.2).

Applesoft Tutorial. Cupertino, Calif.: Apple Computer, Inc., 1982.

Leventhal, Lance. *6502 Assembly Language Programming.* Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

Synertek Hardware Manual. Santa Clara, Calif.: Synertek Incorporated, 1976.

Does not contain instructions new to 65C02, but is the only currently available manufacturer's hardware manual for 6500 series microcomputers.

Synertek Programming Manual. Santa Clara, Calif.: Synertek, Incorporated, 1976.

The only currently available manufacturer's programming manual for 6500 series microcomputers.

Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte* Vol. 5, No. 11 (November, 1980).

———. "A Simplified Theory of Video Graphics, Part II." *Byte* Vol. 5, No. 12 (December, 1980).

———. "More Colors for Your Apple." *Byte* Vol. 4, No. 6 (June, 1979).

———. "True Sixteen-Color Hi-Res." *Apple Orchard* Vol. 5, No. 1 (January, 1984).

Wozniak, Steve: "System Description: The Apple II." *Byte* Vol. 2, No. 5 (May, 1977).

Index

References to entries in Volume 2 are in square brackets [ ].

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

030-1022-A

Printed in U.S.A.